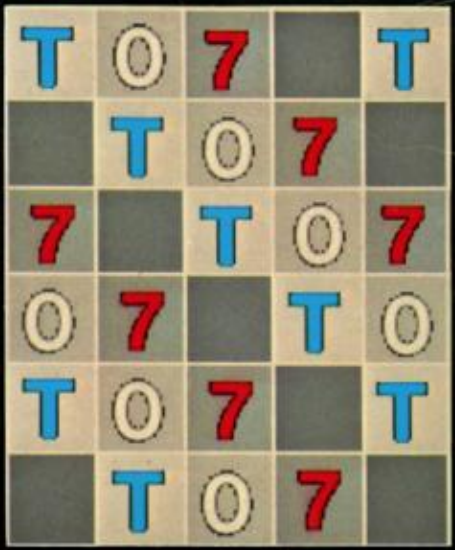
M. OURY

MAITRISEZ LE TO 7

DU BASIC AU LANGAGE MACHINE



MICRO SYSTEMES

ETSF

Collection
MICRO-SYSTEMES
dirigée par ALAIN TAILLIAR
directeur de la rédaction de MICRO-SYSTEMES

MAITRISEZ LE TO 7 du Basic au langage machine

COLLECTION E.T.S.F. MICRO-SYSTEMES

1 – Un microprocesseur pas à pas
2 – Systèmes à microprocesseur
3 – Maîtrisez votre ZX 81
4 – Du Basic au Pascal
5 – Vous avez dit Basic ?
6 – Vous avez dit micro ?
7 – Pilotez votre ZX 81
8 – La micro-informatique et son ABC
9 – Maîtrisez le TO 7

COLLECTION POCHE-INFORMATIQUE

1 – 50 programmes pour ZX 81
2 – Montages périphériques pour ZX 81
3 – Passeport pour Applesoft
4 – Passeport pour Basic
5 – Mathématiques sur ZX 81
6 – Passeport pour ZX 81
7 – 50 programmes pour Casio FX-702 P et FX-801 P

Illustration de couverture : PRO. Publicité et réalisations audiovisuelles - Paris

« La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que « les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1er de l'Art. 40). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée per les Art. 425 et suivants du Code pénal ».

© 1983 – E.T.S.F.

ISBN 2-85535-060-3

ISSN 0759-9498

Michel OURY

MAITRISEZ LE TO 7
du Basic au langage machine

Diffusion
EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES
2 à 12, rue de Bellevue, 75940 Paris Cedex 19

Page blanche

Sommaire

PREMIÈRE PARTIE

Le système TO 7

Introduction 9

Chapitre I. **Le matériel** 11
– Le microprocesseur 11
– Les mémoires 12
– Les entrées/sorties : 12
● écran 12
● clavier 15
● crayon optique 18

Chapitre II. **Le logiciel** 19
– Le moniteur 19
– Le Basic Microsoft 31
– Les micro-didacts, les jeux 31

Chapitre III. – **Les extensions** 31
– Le lecteur de disquettes 31
– La mémoire 32
– L'imprimante 32
– L'interface série RS 232 32
– L'extension jeux/convertisseur D/A 32
– Le codeur-modulateur (SECAM ou PAL) 33
– L'extension Télétel 33
– L'extension Antiope 33
– L'extension de synthèse vocale 33

Le Basic TO 7

Chapitre IV. **Les bases du langage** 35
- Exemple introductif 35
- Les différents types de variables 36
- Les instructions d'entrée/sortie : 37
  INPUT, LINE INPUT, INPUT WAIT, INPUT$(N), INKEY$ 37
  READ, DATA, RESTORE 40
  PRINT, PRINT TAB, PRINT USING 42
- Applications 45

Chapitre V. **La gestion de l'écran** 45
- Exemple introductif 46
- Les instructions de définition : ATTRB, UNMASK, SCREEN, COLOR, CONSOLE 47
- Les instructions de positionnement : LOCATE, PSET, LINE, POS, CSRLIN, SCREEN (X,Y) 52
- Applications 54

Chapitre VI. **Les caractères utilisateurs** 55
- Exemple introductif 58
- Les instructions de définition : DEFGR$(X), CLEAR,,N 59
- Applications 59

Chapitre VII. **Branchements, itérations et sous-programmes** 62
- Exemple introductif 62
- Les branchements inconditionnels : GOTO, ON GOTO 65
- Les branchements conditionnels : IF... THEN... ELSE (AND, OR, NOT, >, <, =) 65
- Les itérations : FOR... TO... STEP, NEXT 67
- Les branchements à des sous-programmes : GOSUB, RETURN, EXEC, DEFUSR, USR, CLEAR, AD 68
- Applications 72

Chapitre VIII. **Les variables numériques** 73
- Représentation interne 73
- Opérateurs arithmétiques (↑, +, − ...) 76
- Opérateurs relationnels (=, <, >, ...) 76
- Fonctions arithmétiques et algébriques 77
- Applications 78

Chapitre IX. **Les chaînes de caractères** 82
- Représentation interne 82
- Opérateur concaténation : + ... 83
- Opérateurs relationnels 83
- Les instructions de chaînes : LEFT$, RIGHT$, MID$, LEN, STR$, VAL, ASC, CHR$, INSTR, HEX$, FRE, MID$( ), CLEAR, OCT$ 85
- Police de caractéres (CHR$) 88
- Codes spéciaux (CHR$) 89
- Caractères semi-graphiques Télétel 90

Chapitre X. **Les tableaux de variables** 92
- Dimensionnement-occupation 92
- Exemples d'applications 94

Chapitre XI. **Les instructions de périphériques** 96
- Le light-pen : PEN, ONPEN GOSUB, ONPEN GOTO, INPUTPEN X,Y, INPEN X,Y, PTRIG 96
- Les manettes de jeux : STICK(N), STRIG(N) 99
- La musique : BEEP, PLAY 100
- Exemples d'applications 101
- Les instructions de fichier : description des fichiers, types de périphériques : LIST, SAVE, SAVEM, CLOSE, OPEN, LOAD, MERGE, LOADM, INPUT#, PRINT#, MOTOR ON,MOTOR OFF, SKIPF, POS, WAIT 103

Chapitre XII. **Les autres fonctions** 112
- Ecriture et lecture en mémoire : PEEK, POKE, VARPTR 112
- Gestion des erreurs : ERR, ERL, ERROR,ON ERROR GOTO, RESUME 116
- Les commandes Basic : AUTO, CONT, DELETE, LIST, NEW, RUN, GOTO, SAVE, TRON, TROFF 118

TROISIÈME PARTIE

Dessins et graphiques

Chapitre XIII. **Les zones graphiques en RAM** 121

Chapitre XIV. **Les fonctions graphiques** 124
- BOX, BOXF 124
- LINE 124

- PSET 124
- POINT 124
- COLOR et SCREEN 125

Chapitre XV. **Applications** 125

QUATRIÈME PARTIE

Le microprocesseur 6809

Chapitre XVI. **Caractéristiques générales** 129

Chapitre XVII. **Structure interne** 130
- Les accumulateurs A, B, D 130
- Les registres S, U, X, Y 130
- Le registre compteur programme PC 131
- Le registre de page DP 131
- Le registre de codes condition CCR 132
- Notion de pile 134

Chapitre XVIII. **Les modes d'adressage** 136
- Adressage inhérent 136
- Adressage immédiat 137
- Adressage étendu 139
- Adressage direct 139
- Adressage étendu indirect 141
- Adressage relatif court 143
- Adressage relatif long 145
- Adressage indexé 145
- Adressage indexé indirect 151
- L'instruction LEA 151

Chapitre XIX. **Le jeu d'instruction** 153

Chapitre XX. **La programmation en Assembleur** 157
- Principaux conseils 157
- Programme de création 158
- Programmes d'applications 159

Annexes 1 à 11 164

Introduction

J'ai voulu dans cet ouvrage ouvrir le micro-ordinateur TO 7 aussi bien au débutant de la première heure qui se penchera principalement sur le langage Basic et les exemples de programmation qui accompagnent la présentation de ce dernier, qu'à celui qui, ayant déjà programmé, veut mieux connaître sa machine.

C'est pourquoi le premier chapitre, qui présente le matériel, ne se contente pas d'une simple description de l'écran, du clavier,... mais donne l'implantation mémoire de ces périphériques et leur fonctionnement, ou plutôt leur gestion par le microprocesseur 68Ø9.

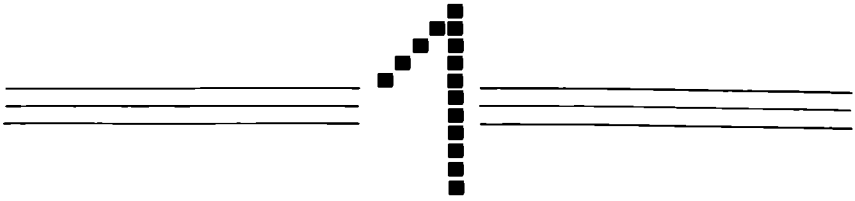
C'est pourquoi également le second chapitre sur le Basic reprend simplement la description des commandes, mais s'étend sur les implantations mémoire des variables, du graphique utilisateur, etc.

C'est pourquoi, enfin, un chapitre spécial est réservé au microprocesseur 68Ø9 pour ceux qui veulent commander le système en « temps réel », c'est-à-dire à la vitesse du microprocesseur, sans être ralentis par l'interpréteur « Basic ». Ils pourront ainsi découvrir la puissance de ce micro haut de gamme, de ses multiples registres échangeables et de son jeu d'instructions symétriques.

Pour pouvoir faire des programmes de jeux, de mesures, de surveillance de processus, d'automatisation des tâches, un apprentissage de la programmation en assembleur termine ce livre qui, je l'espère donnera ainsi satisfaction à tous.

Michel OURY

Page blanche



Le système TO 7

CHAPITRE I **Le matériel**

I. Le microprocesseur

Le système TO 7 est organisé autour d'un des microprocesseurs les plus performants actuellement sur le marché des 8 bits, puisqu'il s'agit du 6809 (voir annexe 1).

Ce microprocesseur sera étudié en détail dans la quatrième partie de l'ouvrage, mais on peut d'ores et déjà rappeler qu'il s'agit d'un « pseudo 16 bits », dont l'organisation logicielle est symétrique et qui possède les modes d'adressage classiques y compris l'étendu indirect et l'indexé indirect, avec la possibilité supplémentaire de réaliser des branchements relatifs longs.

Ce microprocesseur surpuissant de la troisième génération est entièrement compatible avec le 6800 et tous les circuits de cette famille, tant sur le plan matériel que logiciel (compatibilité ascendante).

Il est donc idéal pour la réalisation de programmes en langage évolué.

Dans le TO 7, il est relié à une horloge externe dont l'oscillation est créée par un quartz de 4 MHz.

II. Les mémoires

– **Les mémoires mortes** du TO 7 sont de plusieurs types :

Dans le PIA 6846 (voir annexe 3) se trouve une ROM de 2 K-octets programmée par masque à la fabrication. Dans cette ROM, de l'adresse &HE800 à l'adresse &HEFFF, se trouve une partie du moniteur. Les 4 K-octets restants se trouvent dans une mémoire morte du type 2532. Le moniteur occupe donc les 6 K-octets de &HE800 à FFFF.

Le décodage des adresses de cette ROM est fait par deux circuits « double décodeur de 1 parmi 4 » (fig. I-1) type 74LS139, de même que pour la ROM qui se trouve dans la cartouche chargeable « mémo 7 », celle qui contient le Basic ou d'autres langages ou jeux.

– **Les mémoires vives** utilisateur ou extension sont des mémoires dynamiques (donc à rafraîchissement) du type 4116, ainsi que les mémoires d'écran.

Le décodage d'adresse du système de mémoires vives est réalisé à l'aide d'une PROM à fusible du type 6331-1 adressée par les bits d'adresse A_{13}, A_{14}, A_{15} en provenance du 6809 ainsi que par le bit P_0 du port d'entrée/sortie du 6846 (fig. I-2).

Le rafraîchissement des mémoires vives est fait alternativement par le 6809 puis par le système de gestion d'écran.

L'adressage de ces mémoires est multiplexé.

III. Les entrées/sorties

Les entrées/sorties standards sont :
- l'écran ;
- le clavier ;
- le crayon optique ;

auxquelles on ajoute rapidement le lecteur/enregistreur à cassettes magnétiques.

– **L'écran** travaille en permanence en mode graphique. On peut donc mixer graphisme et caractères. La résolution est de 64000 points (320 × 200). Les points sont groupés par 8 points horizontaux. Dans chaque groupe, on ne peut définir que 2 couleurs choisies parmi les 8 possibles (noir, rouge, jaune, vert, bleu, magenta, cyan et blanc). On peut y écrire tous les caractères ASCII, les caractères Télétel et définir ses propres caractères (maximum 128).

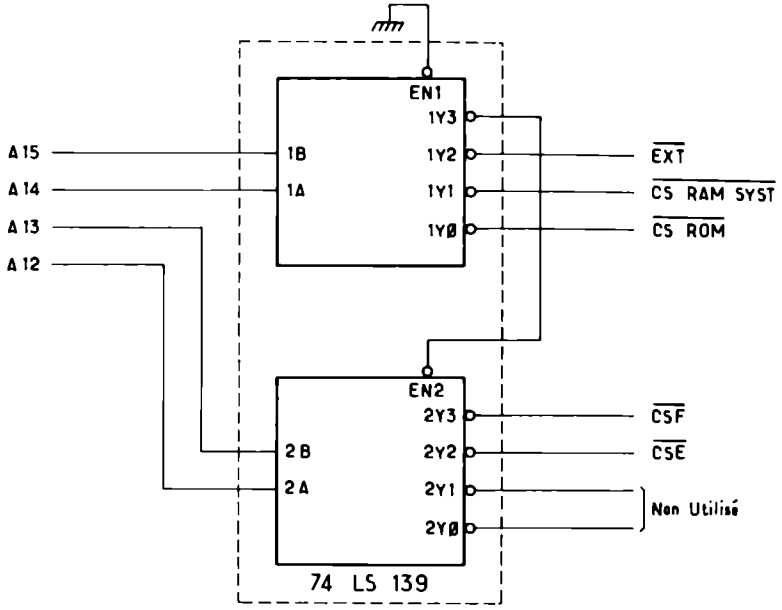


Fig. I. 1. – Décodage des zones mémoires.

Adresses	Capacité	Sortie	Commentaires
0000 à 3FFF	16 K	CS ROM	Validation de la cartouche mémo 7
4000 à 7FFF	16 K	CS RAM SYST	Inhibitions de données
B000 à BFFF	16 K	EXT	Validation partielle de la RAM extension
C000 à CFFF D000 à DFFF	8 K	CSC CSD	Non utilisé
E000 à EFFF	4 K	CSE	Validation partielle
F000 à FFFF	4 K	CSF	Validation de la ROM moniteur

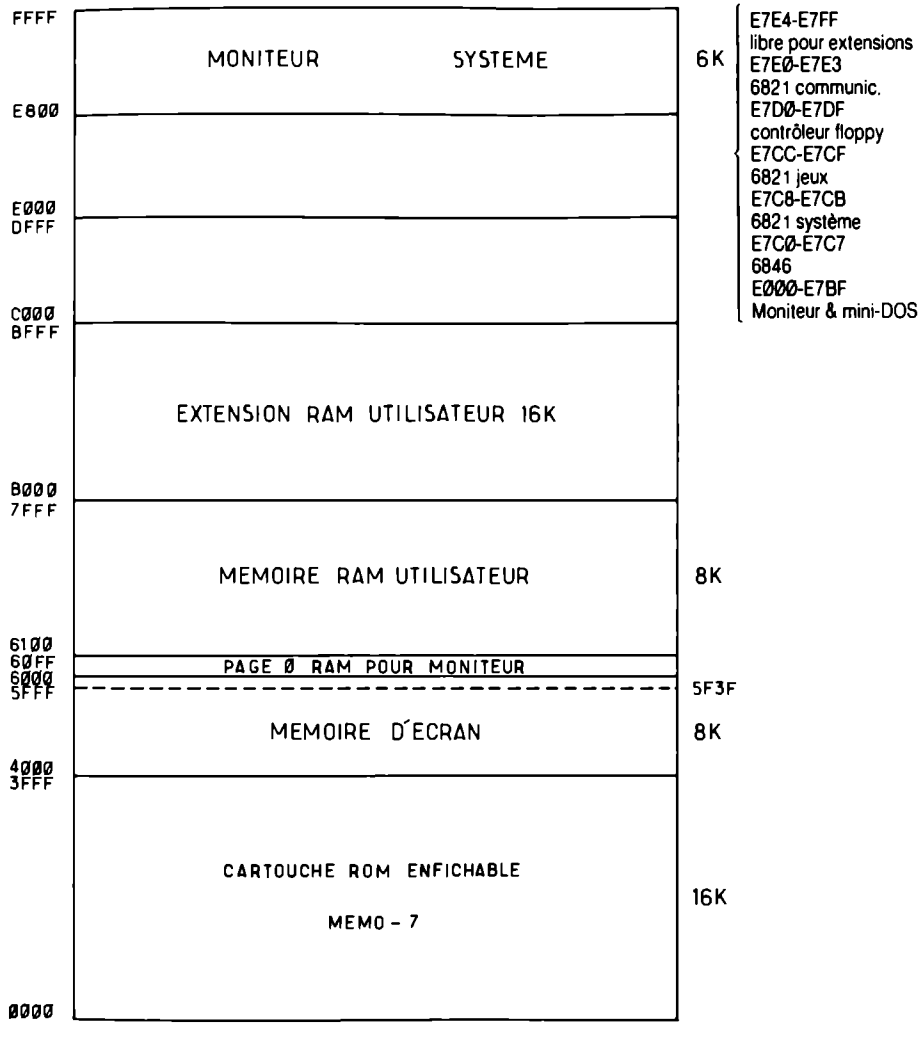


Fig. I.2. – Architecture générale de la mémoire TO 7.

L'alphabet contient les majuscules, les minuscules ainsi que les accents, le « ç » et les symboles spéciaux les plus courants. On peut doubler la hauteur et/ou la largeur de tous les caractères.

Pour la gestion de l'écran, on se reportera aux chapitres « Gestion de l'écran », « Les caractères utilisateurs » ainsi que « Dessins et graphiques ».

– **Le clavier** est du type plat à membrane. Il comporte 57 touches dont les touches de fonctions suivantes :

[STOP] Cette touche arrête le défilement de l'écran, qui reprend par une action sur n'importe quelle autre touche (équivalant à CTRL B).

[CTRL] Cette touche de contrôle doit être associée à une autre touche, pressée simultanément, et dont elle modifie ainsi le sens.

CTRL-C arrête le programme.

CTRL-X efface la fin de la ligne à partir de la position courante du curseur.

[RAZ] Cette touche efface l'écran et ramène le curseur en haut et à gauche (équivalant à CTRL-L).

[●] Le point jaune donne accès aux caractères marqués en jaune sur les touches. Enfoncée en même temps que la barre d'espace, cette touche permet le passage en minuscules.

Dans le mode minuscules, la diode électroluminescente MIN s'allume et alors l'action combinée et simultanée sur le point jaune et sur une autre touche permet de taper une majuscule.

[ACC] Cette touche permet le passage en mode accentué. La procédure doit être alors la suivante :

- taper l'accent ;
- taper la lettre à accentuer.

On obtient les accents :

- grave par action sur « point jaune » et « Ø » ;
- aigu par action sur « point jaune » et « 7 » ;
- circonflexe par action sur « point jaune » et « @ » ;
- tréma par action sur CTRL et ● et @ ;

et la cédille par action sur le C seul.

Pour taper des lettres accentuées, on peut utiliser les codes ASCII correspondant :

- à la procédure d'accentuation → 22_{10}
- à l'accent grave → 65_{10}
  aigu → 66_{10}
  circonflexe → 67_{10}
  tréma → 72_{10}
  cédille → 75_{10}
- à la lettre minuscule que l'on veut accentuer.

Exemple : pour écrire ç, on peut faire :

```
PRINT CHRS$(22)+CHR$(75)+CHRS$(99)
```

accent c

[ENTREE] Cette touche valide une réponse, une commande, une ligne de programme, etc.

[↑] [↓] [←] [→] Ces quatres touches permettent le déplacement du curseur dans les quatre directions, sans modification de l'écran (donc sans écrire dans la mémoire d'écran).

[INS] Cette touche crée un espace là où se trouve le curseur afin d'y insérer un caractère.

[EFF] Cette touche efface le caractère courant et provoque un décalage gauche de toute la fin de la ligne.

Toutes les touches précédentes ont un code ASCII. Elles peuvent donc être considérées comme des caractères et être appelées par des instructions CHR$( ) ou par un sous-programme en assembleur.

– **Gestion du clavier :** le clavier est géré par le PIA système du type 6821 (voir annexe 2) placé aux adresses &HE7C8 à &HE7CB. Le clavier de 58 touches (57 effectives car le point jaune est en double) est organisé selon une matrice de 8 × 8 (fig. I-3).

Le port A du 6821 est programmé en entrée, alors que le port B est programmé en sortie (fig. I-4).

Quand aucune touche n'est enfoncée, toutes les entrées du PORT A sont au 1 logique, car tirées au + 5 V par des résistances. Le port B fait passer l'un après l'autre les bits BØ, B1,..., B7 au Ø logique.

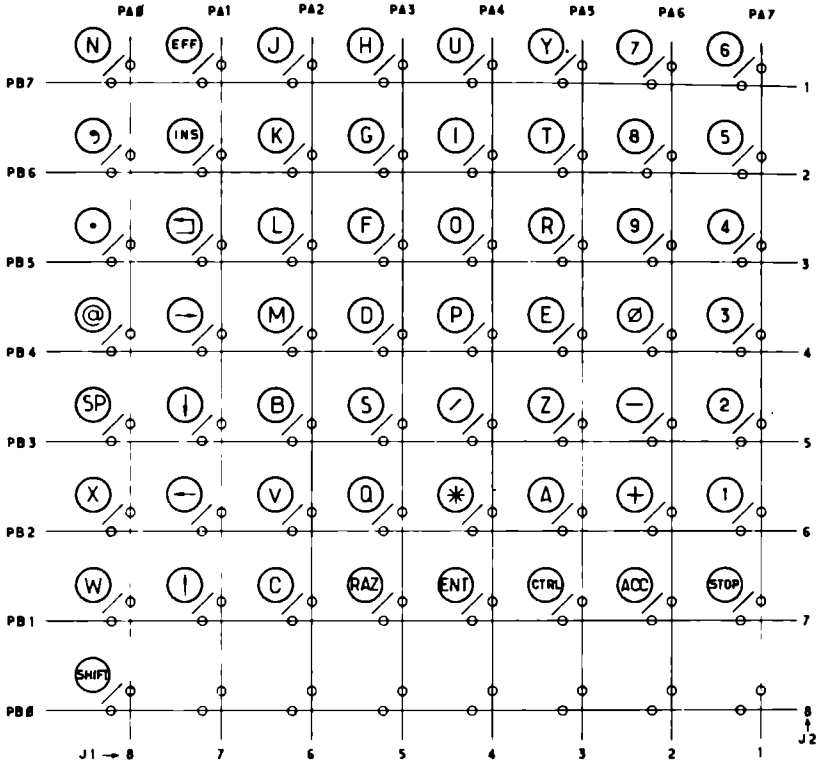


Fig. I.3. – Organisation matricielle du clavier.

Si, à un instant donné, une touche est enfoncée, la valeur lue sur le PORT A diffère de &HFF (255_{10}). Un sous-programme de décodage enregistre alors cette valeur, ainsi que celle présente en sortie sur le PORT B, la combinaison des deux permettant de savoir que la touche enfoncée se trouve à l'intersection de la colonne du bit du Port A à Ø et de la ligne du bit du Port B à Ø.

Exemple : si, au moment où une touche est enfoncée, on a :

(PORT A) = &HF7
(PORT B) = &HFD

la touche enfoncée est RAZ

On peut ainsi détecter l'action simultanée sur deux touches. C'est la routine « GETC$ » qui permet le transcodage en caractères ASCII avant envoi vers la mémoire d'écran.

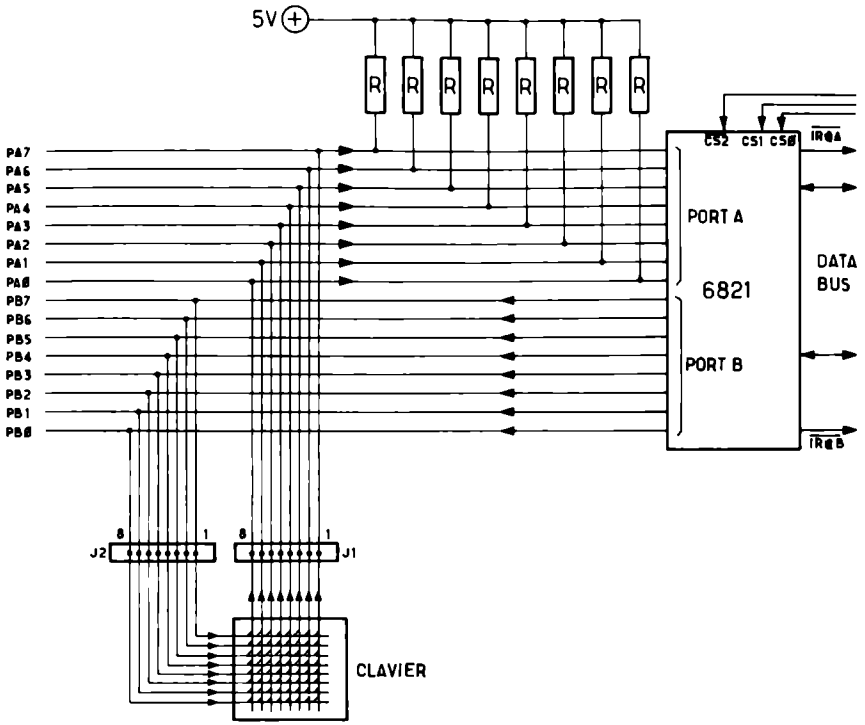


Fig. I.4. – Gestion du clavier.

– **Le crayon optique :** en fonctionnement « crayon optique », le moniteur commande la programmation du CB2 du 6821 en sortie afin de supprimer la surveillance du clavier. Le PORT A et le PORT B sont ensuite programmés en entrée afin de recevoir les informations en provenance du crayon optique.

Le balayage du spot sur l'écran correspond à la recopie de la mémoire d'image. A un instant donné de départ, la première case de mémoire d'écran &H4000 est lue, et l'octet qu'elle contient permet l'allumage ou non dans la couleur choisie des huit premiers points de la première ligne graphique de l'écran. Puis la case mémoire suivante permet l'écriture du groupe point ligne suivant, etc.

A tout instant, il y a concordance entre l'octet écran et la case mémoire, donc le 6809 connaît à tout instant l'adresse de l'octet recopié sur l'écran.

Grâce au phototransistor placé en bout de crayon, le 68Ø9, via le PIA 6821, saura quelle est l'adresse mémoire du groupe visé par le crayon, puisque chaque octet écran est écrit seul pendant que les autres sont éteints. En effet seule la persistance rétinienne nous donne l'illusion d'un écran complètement allumé, alors que les points ne sont éclairés que 8 par 8, les uns après les autres.

Suivant la distance du crayon à l'écran, plusieurs groupes sont envoyés au TO 7 qui à l'aide d'une routine fait le tri et ne conserve que le groupe **le plus à droite** parmi tous les groupes contigus détectés.

CHAPITRE II **Le logiciel**

Les exemples qui suivent en langage machine peuvent être testés à l'aide du programme de création page 158.

I. Le moniteur

C'est un programme en langage machine, implanté dans la mémoire morte de l'ordinateur et dont le rôle est de gérer les échanges avec les interfaces de périphériques.

Ce programme contient une vingtaine de sous-programmes que l'utilisateur peut appeler par EXEC en Basic ou JSR en langage d'assemblage. Pour pouvoir faire fonctionner convenablement ces routines, il est nécessaire de définir des paramètres (constantes ou variables), soit en les plaçant dans certains registres du 68Ø9 (accumulateur B, registres d'index X, Y, etc.), soit en les plaçant en mémoire vive à des adresses connues. Dans ce dernier cas, on dira que le paramètre est placé dans un registre moniteur.

L'ensemble des registres du moniteur est en RAM de l'adresse &H6ØØØ à l'adresse &H6ØFF et correspond à ce que l'on nomme la page Ø du moniteur système.

On trouvera la liste de toutes les adresses de la page Ø à l'annexe 4.

Les sous-programmes de gestion des interfaces de périphériques sont les suivants :

- Gestion de l'écran alphanumérique → PUTC$, GETC$, INIT$, CHPL$
  écran graphique → PLOT$, DRAW$, GETP$
- Gestion du clavier → GETS$, KTST$
- Gestion de la cassette → K7CO$
- Gestion de l'interface de communication → RSCO$
- Gestion du générateur musical → NOTE$
- Gestion du light-pen → LPIN$, GETL$
- Gestion du lecteur de disquettes → DKCO$, DKBO$, FORM$
- Gestion des manettes de jeux → JOYS $

Le moniteur contient également des routines d'usage général, telle que l'intialisation, le reset des attributs d'écran, et le maniement d'interruptions variées :

- Point d'entrée à chaud → MENU$
- Interrupteur du clavier et du curseur → KBIN$

Les adresses des points d'entrée dans ces sous-programmes sont données en annexe 5.

Etude de PUTC$ (&HE8Ø3)

C'est le programme de gestion alphanumérique de l'écran qui interprète directement la syntaxe Télétext et contrôle totalement la structure de l'écran. En plus de la gestion des caractères courants, il définit les attributs d'écran : mode page/scrolling, couleur...

Il contient également les commandes de la position du curseur ainsi que son état (éteint ou allumé).

Tous les arguments sont passés à l'aide de l'accumulateur B du 68Ø9.

– *Définition de la fenêtre*

Ligne basse : D_BU_B exemple D_BU_B = 21
Ligne haute : D_HU_H exemple D_HU_H = Ø4

La procédure est la suivante (ligne basse) :

```
LDB #$US          C6 1F
JSR PUTC$         BD E8Ø3
LDB #$1DB         C6 12
JSR PUTC$         BD E8Ø3
LDB #$1UB         C6 11
JSR PUTC$         BD E8Ø3
```

Puis (ligne haute)

```
LDB #$US          C6 1F
JSR PUTC$         BD E8Ø3
LDB #$2DH         C6 2Ø
JSR PUTC$         BD E8Ø3
LDB #$2UH         C6 24
JSR PUTC$         BD E8Ø3
```

– *Gestion du curseur*

Une procédure du type suivant :

```
LDB #$XX          C6 XX
JSR PUTC$         BD E8Ø3
```

Permet d'obtenir :

- si XX = 11 (DC1) l'allumage du curseur
- si XX = 14 (DC4) l'extinction du curseur
- si XX = ØD (CR) le retour en début de ligne
- si XX = ØC (FF) le retour en début de fenêtre avec effacement de la fenêtre
- si XX = 1E (RS) le retour en début de fenêtre
- si XX = Ø8 (BS) le retour arrière d'une position
- si XX = Ø9 (HT) l'avance d'une position
- si XX = ØA (LF) la descente d'une ligne
- si XX = ØB (VT) la montée d'une ligne
- si XX = 18 (CAN) l'effacement de la ligne à partir du curseur sans modifier sa position
- si XX = 2Ø (SP) l'effacement du caractère, remplacé par la couleur FOND
- si XX = 7F (DEL) l'effacement du caractère, remplacé par la couleur FORME

En utilisant la commande US (&H1F), on peut placer le curseur à un endroit défini par ses coordonnées L,C. La procédure sera la suivante :

LDB #$US/JSR PUTC$/LDB #$ (&H4Ø+L)/JSR PUTC$/
LDB #$(&H4Ø+C)/JSR PUTC$.

– *Définition d'un attribut*

Deux appels de PUTC$ sont nécessaires :

– La commande ESC (&H1B) est chargée dans B puis appel 1.
– L'attribut est chargé dans B, puis appel 2 de PUTC$.

L'attribut définit en général la couleur du CADRE, du FOND ou de la FORME.

Au cadre correspond la valeur &HØØ
Au fond correspond la valeur &H5Ø
A la forme correspond la valeur &H4Ø

Les couleurs sont :

NOIR	ROUGE	VERT	JAUNE	BLEU	MAGENTA	CYAN	BLANC
Ø	**1**	**2**	**3**	**4**	**5**	**6**	**7**

Pour définir le CADRE en BLEU, l'attribut sera &H64.
Pour définir la FORME en BLEU, l'attribut sera &H44.
Pour définir la FORME en JAUNE, l'attribut sera &H43.
Pour définir le FOND en BLANC, l'attribut sera &H57.

Exemple : définition du cadre en bleu.

```
LDB  #$ESC                 C6  1B
JSR  PUTC$                 BD  E8  Ø3
LDB  #$ (CADRE + BLEU)     C6  64
JSR  PUTC$                 BD  E8  Ø3
```

L'attribut permet également la définition de la dimension du caractère :

– double hauteur attribut &H4D
– double largeur attribut &H4E
– double hauteur et double largeur attribut &H4F
– dimensions normales attribut &H4C

De même, l'attribut &H5C permet l'inversion vidéo, l'attribut &H58 permet le masquage et l'attribut &H5F le démasquage.

Si on veut faire porter l'attribut sur toute la fenêtre, il faudra quatre appels successifs de la routine PUTC$, qui enverront successivement, par chargement de B :

ESC/ # /SP/attribut, soit, en hexa : 1B/23/2Ø/attribut.

L'attribut &H6A correspond au scroll normal, l'attribut &H6E au scroll lent, et l'attribut &H6B ramène au mode page en réinitialisant certains attributs.

– *Affichage d'un caractère standard*

Tout caractère ayant un code ASCII peut être affiché à la position courante du curseur, avec les attributs définis auparavant, en chargeant l'accumulateur B avec la valeur hexadécimale du code ASCII du caractère, puis en appelant PUTC$.

Exemple : on veut afficher « Paul » dans le mode courant.

P → &H5Ø, a → &H61, u → &H75, l → &H6C.

```
Donc : LDB #$«P»     C6  5Ø
       JSR PUTC$     BD  E8  Ø3
       LDB #$«a»     C6      61
       JSR PUTC$     BD  E8  Ø3
       LDB #$«u»     C6  75
       JSR PUTC$     BD  E8  Ø3
       LDB #$«l»     C6  6C
       JSR PUTC$     BD  E8  Ø3
```

– *Affichage d'un caractère spécial*

En général, ce caractère sera un accent ou la cédille. Dans ce cas trois appels de PUTC$ sont nécessaires :

- Lors du premier appel, l'accumulateur B contient le code de la séquence spéciale SS2 (&H16).
- Lors du deuxième appel, l'accumulateur B contient le code du signe spécial (accent), soit : &H41 accent grave
  &H42 accent aigu
  &H43 accent circonflexe
  &H48 tréma
  &H4B cédille
- Lors du troisième appel, B contient le code de la lettre à accentuer.

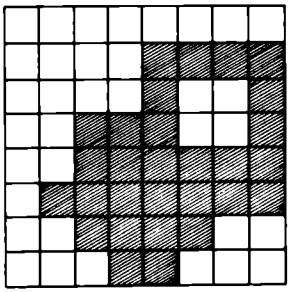
Parmi les signes spéciaux figurent également :

- le degré ° .... &H3Ø
  ← &H2C
- les flèches ↑ &H2D
  → &H2E
  ↓ &H2F
- le signe ± &H31
- l'« Œ » majuscule &H6A
- l'« œ » minuscule &H7A

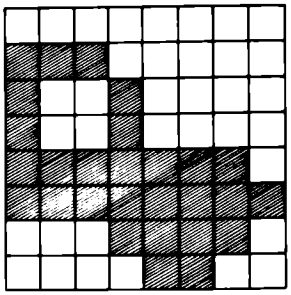
- ***Affichage d'un caractère graphique utilisateur***

Comme nous le verrons au chapitre « Les caractères utilisateurs », chacun peut définir jusqu'à 128 caractères. Chaque caractère est formé de 8 segments de 8 points (matrice 8 × 8). A chaque segment correspond une valeur numérique hexadécimale obtenue en faisant correspondre à un point du fond le bit Ø et à un point de la forme le bit 1.

La petite voiture constituée de deux caractères que l'on dessine un peu plus loin est codée comme suit :



1er caractère = 18, 3C, 7F, 3F
(N° Ø) 39, Ø9, ØF, ØØ



2e caractère = ØC, 1E, FF, FE,
(N° 1) 9Ø, 9Ø, EØ, ØØ

L'ensemble de tous les caractères définis par l'utilisateur est rangé en mémoire à une adresse connue, par exemple &H7ØØØ. Ce qui donne l'occupation mémoire ci-dessous.

TABLE GRAPHIQUE « VOITURE »

ADRESSE hexa	CONTENU hexa
7ØØØ	18
7ØØ1	3C
7ØØ2	7F
7ØØ3	3F
7ØØ4	39
7ØØ5	Ø9
7ØØ6	ØF
7ØØ7	ØØ

ADRESSE hexa	CONTENU hexa
7ØØ8	ØC
7ØØ9	1E
7ØØA	FF
7ØØB	FE
7ØØC	9Ø
7ØØD	9Ø
7ØØE	EØ
7ØØF	ØØ

Chaque caractère a un code ASCII. Le premier caractère a pour code &HØØ, le second &H81, ... et le 128e, &HFF.

Pour afficher un caractère graphique, il faut :

- charger le registre X avec l'adresse de la table mémoire,
- ranger X dans le pointeur USERAF (&H6ØØD),
- charger B avec le code ASCII du caractère,
- appeler PUTC$.

Exemple : pour afficher la voiture, on écrira :

```
LDX  #$TABLE 8E  7Ø  ØØ
STX  USERAF  BF  6Ø  2D
LDB  #$8Ø    C6  8Ø
JSR  PUTC$   BD  E8  Ø3
LDB  #$81    C6  81
JSR  PUTC$   BD  E8  Ø3
```

Etude de INIT$ (&HE8ØØ)

Ce sous-programme réinitialise la fenêtre en mode plein écran, le mode SCROLL et CARACTERE avec couleur.

Etude de PLOT$ (&HE8ØF)

Ce sous-programme permet d'afficher **un point graphique** défini par ses coordonnées ligne L et colonne C sur l'écran.

Procédure :

- Charger le registre X avec C (Ø ⩽ C ⩽ 319)
- Charger le registre Y avec L (Ø ⩽ L ⩽ 199)
- Appeler PLOT$.

La couleur du point (de Ø à 7 pour la FORME, de – 1 à – 8 pour le FOND) doit être rangée dans le registre moniteur FORME qui se trouve page zéro du moniteur (&H6Ø38). Le registre CHDRAW doit être préalablement mis à Ø. CHDRAW se trouve page zéro (&H6Ø41).

Etude de CHPL$ (&HE833)

Ce sous-programme permet d'afficher **un caractère** défini par ses coordonnées C (1 ⩽ C ⩽ 4Ø) et L (Ø ⩽ L ⩽ 24).

Procédure :

- Charger le registre X avec C
- Charger le registre Y avec L
- Mettre le code ASCII du caractère dans le registre CHDRAW
- Appeler CHPL$.

Etude de DRAW$ (&HE8ØC)

Ce sous-programme permet de tracer en mode graphique, ou caractère, une droite allant de la position courante du curseur jusqu'au point défini par ses coordonnées C et L.

Suivant que l'on est en mode graphique ou caractère, on respectera les initialisations conformément à ce qui a été décrit pour PLOT$ ou CHPL$.

On charge X avec C, Y avec L, on met éventuellement le caractère dans le registre CHDRAW (ou bien on y met Ø) puis on appelle DRAW$.

Note : Un « bug » dans la version 1 du moniteur détruit le contenu du registre U lors d'un tracé *horizontal.*

Etude de KTST$ (&HE8Ø9)

Ce sous-programme permet de savoir si une touche du clavier a été appuyée, mais il n'indique pas quelle touche l'a été. L'indication se fait par l'intermédiaire du bit de retenue C du registre de CODES CONDITIONS du 68Ø9 (voir ce chapitre).

- Aucune touche appuyée ⇒ C = Ø
- Une touche appuyée ⇒ C = 1.

Etude de GETC$ (&HE8Ø6)

Ce sous-programme ramène dans l'accumulateur B le code ASCII de la touche appuyée. Si aucune touche n'est appuyée, B = Ø.

Application : détection et affichage en mode courant d'un caractère standard.

```
DETEC  JSR   GETC$  BD E8 Ø6
       TSTB         5D          si B = Ø
       BEQ   DETEC  27 FA
       JSR   PUTC$  BD E8 Ø3
       BRA   DETEC  2Ø F5
```

Ce programme ne pourra être arrêté que par action sur la touche de réinitialisation du TO7.

Etude de LPIN$ (&HE81B)

Ce sous-programme teste l'état du micro contact du light-pen et l'indique dans le bit de retenue C du registre de codes.

Conditions :

- Si C = Ø, le contact n'est pas appuyé
- Si C = 1, le contact est appuyé.

Etude de GETL$ (&HE818)

Si le contact est pressé (C = 1), ce sous-programme ramène dans les registres X et Y les coordonnées C (colonne) et L (ligne) du point détecté.

Ce point sera le point milieu du segment détecté (voir le chapitre sur le light-pen).

Etude de RSCO$ (&HE812)

Ce sous-programme permet l'échange d'informations sous forme série ou parallèle, en écriture ou en lecture, avec des périphériques (par exemple, une imprimante).

Tous les échanges se font par un registre moniteur RSOPC, situé page zéro du moniteur à l'adresse &H6Ø2B. Un autre registre moniteur renseigne sur la bonne marche des échanges, il s'agit de RS.STA situé à l'adresse &H6Ø2C.

Les commandes possibles sont :

- ouverture en lecture/écriture série (RS 232) si RSOPC contient &HØ1
- lecture d'un octet si RSOPC contient &HØ2
- ouverture en écriture seulement (RS 232) si RSOPC contient &HØ4
- écriture d'un octet si RSOPC contient &HØ8
- fermeture si RSOPC contient &H1Ø
- copie graphique sur l'imprimante thermique si RSOPC contient &H2Ø
- ouverture en écriture parallèle si RSOPC contient &H4Ø.

En cas d'erreur dans les échanges (périphérique non prêt, ordre non exécuté, périphérique déjà ouvert...), le bit de retenue du CCR est mis à 1.

Si le périphérique n'est pas prêt, alors le registre moniteur RS.STA contient &HØØ, sinon il indique l'état du périphérique.

L'octet est transmis par l'intermédiaire de l'accumulateur B.

En mode de transmission série (RS 232), il faut en outre définir la vitesse de transmission et le nombre de bits transmis (7 ou 8). A cet effet, nous disposons de deux registres moniteurs : BAUDS (16 bits) en &H6Ø44 et NOMBRE en &H6Ø46.

La correspondance (BAUDS) → vitesse est la suivante :

- (BAUDS) = &HØ46A → vitesse 110 bauds (bits/s)
- (BAUDS) = &HØ19B → vitesse 300 bauds (bits/s)
- (BAUDS) = &HØØCA → vitesse 600 bauds (bits/s)
- (BAUDS) = &HØØ62 → vitesse 1 200 bauds (bits/s)
- (BAUDS) = &HØØ2E → vitesse 2 400 bauds (bits/s)
- (BAUDS) = &HØØ15 → vitesse 4 800 bauds (bits/s)
- (BAUDS) = &HØØØ7 → vitesse 9 600 bauds (bits/s).

La correspondance (NOMBRE) → nombre de bits est la suivante : – (NOMBRE) = &H8Ø → 8 bits transmis
– (NOMBRE) = &H4Ø → 7 bits transmis.

Chaque imprimante ayant en général ses codes opérations spécifiques, il est conseillé d'en prendre connaissance préalablement.

Exemple : impression du prénom « Paul » sur l'imprimante PR 90-080 (imprimante parallèle) :

```
LDA  #$4Ø      86 4Ø        }
STA  $6Ø2B     B7 6Ø 2B     } ouverture en mode parallèle
JSR  RSCO$     BD E8 12     }
LDA  #Ø8       86 Ø8        }
STA  $6Ø2B     B7 6Ø 2B     } écriture d'un caractère
LDB  #$ØF      C6 ØF        }
JSR  RSCO$     BD E8 12     } commande mode caractères
LDB  #$5Ø      C6 5Ø        }  écriture
JSR  RS CO$    BD E8 12     }  « P »
LDB  #$61      C6 61        }
JSR  RSCO$     BD E8 12     }  « a »
LDB  #$75      C6 75        }
JSR  RSCO$     BD E8 12     }  « u »
LDB  #$6C      C6 6C        }
JSR  RSCO$     BD E8 12     }  « l »
LDB  #ØD       C6 ØD        }
JSR  RSCO$     BD E8 12     }  « CR »
LDA  #$1Ø      86 1Ø        }
STA  $6Ø2B     B7 6Ø 2B     } fermeture
JSR  RSCO$     BD E8 12     }
RTS            39           retour de S.P.
```

Attention : ce programme ne contient aucun test de gestion des erreurs pendant l'échange et suppose donc que l'imprimante n'était pas ouverte au moment de l'appel.

On remarquera également qu'il faut faire un « retour chariot » pour vider le buffer d'émission, avant de fermer l'imprimante.

Etude de NOTE$ (&HE81E)

Ce sous-programme permet de jouer une note dont on définit dans chaque registre moniteur correspondant, l'octave, la durée, le temps et le timbre.

Registre OCTAVE (&H6Ø36)

Contient l'octave de &HØØ1Ø (octave la plus basse) à &HØØØ1 (octave la plus haute).

Registre DUREE (&H6Ø33)

Contient &HØØ6Ø pour une ronde
&HØØ3Ø pour une blanche
&HØØ18 pour une noire
&HØØØC pour une croche
&HØØØ6 pour une double croche

Registre TEMPO (&H6Ø31)

Définit le tempo de &HØØØ1 (le plus rapide) à &ØØFF (le plus lent).

Registre TIMBRE (&H6Ø35)

Définit sur un seul octet le timbre de la note, de &HØØ (son continu) à &HFF (son le plus prononcé).

Les notes sont transmises par l'accumulateur B, avant l'appel de NOTE$, avec le code suivant :

PAUSE → &H3Ø	MI → &H35	LA → &H3A
DO → &H31	FA → &H36	LA# → &H3B
DO# → &H32	FA# → &H37	SI → &H3C
RE → &H33	SO → &H38	DO → &H3D
RE# → &H34	SO# → &H39	

Etude de JOY$ (&HE827)

Ce sous-programme permet de connaître l'état du bouton-poussoir et la position du levier d'une des deux manettes de jeu.

L'accumulateur A doit contenir le numéro Ø ou 1 de la manette testée.

La position du levier revient dans l'accumulateur B par une valeur de Ø à 8 correspondant exactement au schéma de la figure XI-3 du chapitre « Les instructions de périphériques » (page 100).

L'état du bouton-poussoir est contenu dans le bit de retenue du registre codes conditions du 6809 :

– Si C = 0 → bouton lâché – Si C = 1 → bouton enfoncé.

L'étude des sous-programmes de gestion du lecteur/ enregistreur de cassettes et du contrôleur de floppy, ne sera pas faite dans cet ouvrage, car ils nécessitent une bonne connaissance des systèmes de codage et leur présentation mérite donc un manuel séparé.

L'ensemble des sous-programmes étudiés dans ce qui précède sera utilisé dans le chapitre XX sur « La programmation en Assembleur ».

II. Le Basic

Seul le moniteur est résident en mémoire morte dans le TO 7.

Les programmes d'application sont donc chargés à l'aide de la cartouche « Mémo 7 » et occupent au maximum les 16 premiers kilo-octets de l'espace mémoire.

Actuellement, le seul langage évolué disponible est le Basic Microsoft 5. Il est écrit en mémoire morte et pourra facilement être échangé avec un « Pascal », « APL » ou autre...

III. Les micro-didacts, les jeux

Deux sortes de logiciels sont actuellement disponibles :

- des logiciels stockés en langage Basic verrouillés sur cassette. Ils ne sont utilisables que si le TO 7 contient la cartouche Mémo 7 « Basic » ;
- des logiciels écrits en langage machine, stockés en mémoire morte dans des cartouches Mémo 7. Ceux-ci ne nécessitent évidemment pas la présence du Basic, qui est d'ailleurs impossible puisqu'également en Mémo 7.

Début 1984 apparaîtront également les fichiers sur disquettes (gestion, paye, fichier-mailing...).

CHAPITRE III **Les extensions**

I. Le lecteur/enregistreur de cassettes

Par l'intermédiaire d'une prise DIN 5 broches, on peut connecter un magnétocassette bipiste au TO 7.

L'une des deux pistes sert à l'enregistrement ou à la lecture des programmes Basic, des fichiers ou des programmes en langage machine.

L'autre piste est une piste audio, reliée via le TO 7 au haut-parleur du téléviseur, et qui permet de suivre éventuellement des explications, commentaires ou des corrections de diction pour les programmes de langues.

II. La mémoire

La mémoire vive du TO 7 peut être portée à 24 K-octets, grâce à un module d'extension de 16 K de RAM dynamique que l'on encarte sur le connecteur arrière. Le brochage de ce connecteur est donné ci-contre (fig. III-1).

III. L'extension communication

Cette extension permet à la fois la liaison avec une imprimante parallèle type Centronics et la liaison série selon le mode RS 232 (fig. III-4).

Cette dernière réalise toutes les fonctions de ce type de communication.

Ce module extension peut se connecter sur l'un quelconque des nez de carte arrière dont le brochage est donné figure III-2.

IV. Le lecteur de disquettes

Cet ensemble est constitué d'un module de contrôle qui peut gérer jusqu'à quatre lecteurs de disquettes.

Les disquettes souples sont du type simple face, simple densité et comprennent 40 pistes de 16 secteurs. Chaque secteur/piste peut contenir 128 octets, ce qui conduit à une capacité de 80 K-octets par disquette.

V. L'extension jeux

Cette extension comporte, outre les manettes digitales, un convertisseur digital/analogique permettant de synthétiser des sons avec quatre voix musicales indépendantes et timbres programmables (fig. III-3).

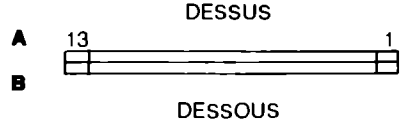
VI. Le codeur-modulateur SECAM

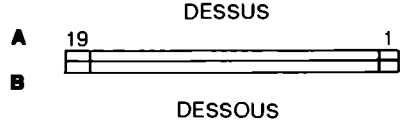
Ce codeur permet le raccordement à un téléviseur couleur via la prise d'antenne, ou à un téléviseur N/B de la même façon.

VII. Les extensions en développement

Une extension de raccordement à Télétel enfichable à l'arrière de l'ordinateur est actuellement en développement. Elle comprend un Modem et une interface de raccordement direct à la ligne PTT conformes aux normes. Elle permet le consultation des banques de données (annuaire électronique, Télétel,... voir annexe 11).

Une extension de réception et de décodage d'Antiope est à l'étude, ainsi qu'un module de synthèse vocale.





	A	B
1	+ 12 V	+ 5 V
2	A_0	D_7
3	A_1	D_6
4	A_2	D_5
5	A_3	D_4
6	A_4	D_3
7	A_5	D_2
8	A_6	D_1
9	R/$\overline{W}$	D_0
10	CLOCK	$\overline{OE}$
11	$\overline{RAS}$	$\overline{CAS}$
12	MASSE	MASSE
13	MASSE	– 5 V

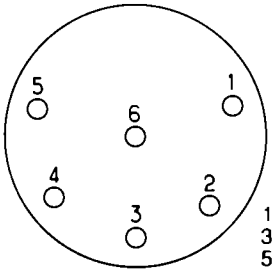
Fig. III-1. – Connecteur pour extension RAM 16 K.

	A	B
1	MASSE	+ 12 V
2		
3	$\overline{E7C}$	Entrée SON
4	$\overline{CSE}$	R/$\overline{W}$
5	A_{11}	E
6	A_{10}	D_7
7	A_9	D_6
8	A_8	D_5
9	A_7	D_4
10	A_6	D_3
11	A_5	D_2
12	A_4	D_1
13	A_3	$\overline{RST}$
15	A_1	$\overline{FIRQ}$
16	A_0	$\overline{NMI}$
17		$\overline{IRQ}$
18		Vidéo
19	+ 5 V	– 5 V

Fig. III-2. – Connecteur extension général.

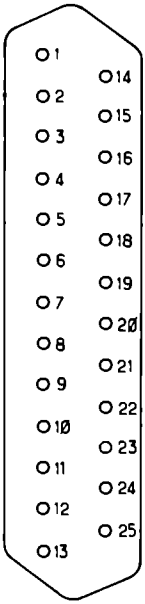
E7C sélectionne E7CX
CSE sélectionne E000-EFFF
Vidéo Pin 19 prise SCART
E extension sélectionnée.

Fig. III-3.
Connecteur manette de jeux.



1. Masse – 2. Avant
3. Arrière – 4. Gauche
5. Bouton-poussoir – 6. Droite.

Fig. III-4.
Prise sortie du contrôleur de communication type CANON 25.



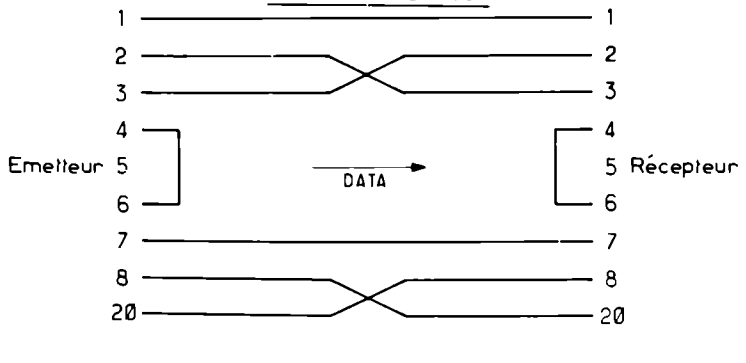
Communication série RS232

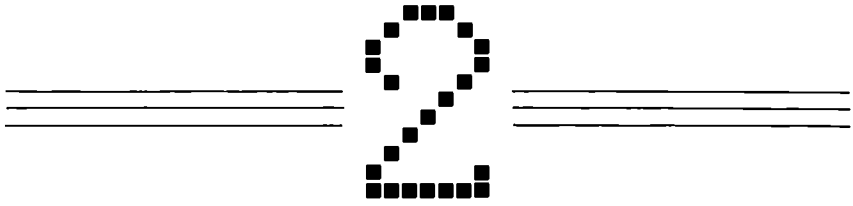
1 : MASSE de protection (blindage)
2 : Transmitted DATA
3 : Received DATA
4 : Request to send
5 : Clear to send
6 : DATA set ready
7 : MASSE signal
8 : Carrier detect
20 : DATA Terminal Ready

Communication parallèle

19 : BUSY
20 : DATA Terminal BUSY
21 : Data 1
9 : Data 2
22 : Data 3
10 : Data 4
23 : Data 5
11 : Data 6
24 : Data 7
12 : Data 8
25 : Acknowledge
13 : Strobe

Liaison entre 2 T07





Le Basic TO 7

CHAPITRE IV **Les bases du langage**

Le langage évolué existant actuellement sur le micro-ordinateur TO 7 est un Basic Microsoft 5, ce qui lui confère une très grande puissance ainsi qu'une grande souplesse d'utilisation. Ce Basic a d'ailleurs été adapté aux particularités du TO 7, aussi bien sur le plan du graphisme que de la musique...

Le TO 7, comme tout micro-ordinateur, est avant tout un organe de communication entre une UNITE CENTRALE et l'UTILISATEUR. Ce dernier entre des informations dans l'unité centrale, principalement à l'aide du clavier, et lit les résultats du traitement de ces informations, principalement sur l'écran.

Ce paragraphe est consacré aux différents moyens utilisables pour entrer des informations dans l'unité centrale et pour afficher les résultats sur l'écran.

1. EXEMPLE INTRODUCTIF

Supposons que l'on souhaite gérer une bibliothèque contenant des romans policiers/livres de fiction/romans historiques/romans d'aventures/livres d'enfants/bandes dessinées.

L'utilisateur de cette « banque de données » doit pouvoir connaître dans chaque série : le nom de l'auteur/le titre de l'ouvrage.

Nous supposerons que la première partie du programme, celle qui présente sur l'écran le titre « BIBLIOTHEQUE » et les différents choix de types d'ouvrages numérotés de 1 à 6, existe déjà. Cette partie sera étudiée dans le chapitre de gestion de l'écran et occupera les lignes de programme de 10 à 110.

Notre programme commence donc ligne 120, et, à cet instant, l'utilisateur a tapé le numéro correspondant au type d'ouvrage choisi.

L'écran doit donc s'effacer, faire apparaître le titre de série des ouvrages choisis, puis la liste numérotée des livres en bibliothèque.

2. LES DIFFERENTS TYPES DE VARIABLE

2.1. Les variables numériques

– Les variables entières sont représentées par un nom de 16 caractères maximum commençant par une lettre, suivi du symbole %.

Exemple : A% ATOME1% sont des variables entières.

– Les variables réelles (ou simple précision) sont représentées par un nom de 16 caractères maximum commençant par une lettre. On peut le faire suivre du symbole !.

Exemple : LOTO RACINE! sont des variables réelles.

– Les variables double précision sont représentées par un nom de 16 caractères maximum commençant par une lettre, suivi du symbole #.

Exemple : X3B# est une variable double précision.

2.2. Les variables chaînes

Une variable chaîne est représentée par un nom de 16 caractères maximum commençant par une lettre et suivi du symbole $. La longueur maximum d'une chaîne est de 255 caractères.

Exemple : LISTE1$ est une variable chaîne.

Pour plus de renseignements, se reporter au chapitre correspondant.

ATTENTION !

Un nom de variable ne doit jamais commencer par un mot du Basic.

Exemple : TOTO est interdit car il commence par TO (voir annexe 8).

3. LES INSTRUCTIONS D'ENTREE/SORTIE

INPUT « Chaîne » ; VAR, VAR...

Cette instruction permet à l'utilisateur de rentrer des informations dans l'unité centrale. Lorsque le moniteur rencontre cette instruction dans un programme, il écrit sur l'écran le commentaire « Chaîne » suivi d'un **?** si le séparateur est le **;** (si on choisi **,** il n'y a pas de **?** après le commentaire), puis il attend la frappe de touches.

Toutes les frappes sont recopiées sur l'écran, jusqu'à ENTREE . A cet instant, l'information est prise en compte et affectée à la première variable de la liste.

Il doit y avoir similitude entre l'information et le type de chaque variable.

On doit ensuite rentrer les valeurs de la deuxième variable, ... Il faut rentrer exactement autant d'informations que INPUT attend de variables.

Si une chaîne de caractères contient un espace en début ou en fin, une virgule..., il faut mettre cette chaîne entre deux guillemets :

Exemple :

```
10 CLS
20 PRINT"     CALCUL DE SURFACE"
30 LOCATE0,5:PRINT"1 - Carre"
40 PRINT"2 - Rectangle"
50 PRINT"3 - Cercle"
60 LOCATE2,10:INPUT"Taper 1,2, ou 3 :",C
70 ON C GOTO100,200,300
80 '
82 '
84 '-------------Surface du carre--------
86 '
88 '
100 CLS:PRINT"SURFACE DU CARRE":LOCATE2,
12
110 INPUT"Longueur du cote :",A:LOCATE2,
14
120 PRINT"Surface :";A*A
130 END
182 '
182 '
```

```
184 '--------------Surface du rectangle--
-
185 '
186 '
200 CLS:PRINT"SURFACE DU RECTANGLE":LOCA
TE2,10
210 INPUT"Longueur des cotes :",A,B:LOCA
TE2,14
220 PRINT"Surface :";A*B
230 END
280 '
282 '
284 '--------------Surface du cercle-----
-
300 CLS:PRINT"SURFACE DU CERCLE":LOCATE2
,10
310 INPUT"Longueur du rayon :",R:LOCATE2
,14
320 PRINT"Surface :";3.14159*R*R
330 END
```

LINE INPUT « Chaîne » ; VAR

Cette instruction permet d'introduire dans l'unité centrale *une chaîne* de 255 caractères maximum y compris les espaces et les symboles.

On peut faire apparaître un commentaire comme pour INPUT.

Il n'y a pas de point d'interrogation.

Exemple :

```
10 CLS
20 LINEINPUT"Donnez votre reponse :",RP$
30 INPUT"Quel est votre prenom :";P$
40 LOCATE4,10
50 PRINTP$+" a repondu "+CHR$(34)+RP$+CH
R$(34)
60 END
```

INPUT WAIT N° ; D, VAR, VAR...

Cette instruction donne un temps limité à D secondes à l'utilisateur pour rentrer les informations correspondant aux variables.

Si la durée D est dépassée, le programme se poursuit à la ligne indiquée par le N°.

Cette instruction est utilisée pour réaliser des temporisations.

Exemple :

```
10 CLS:PRINT"Combien font 13*11 ? :"
20 INPUTWAIT100:5,X
30 IF X=143THENPRINT"BRAVO":END
100 PRINT"Temps depasse.":END
```

INPUT$ (N)

Cette instruction permet de rentrer N caractères du clavier, y compris les caractères de contrôle, sauf CNTRL C, sans écho sur l'écran et sans être obligé de taper ENTREE .

Exemple :

```
10 PRINT"Voulez-vous continuer ?"
20 X$=INPUT$(1)
30 IFX$="O"GOTO40 ELSE END
40 PRINT"Bien continuons"
```

Le programme ci-dessus permet de tester si la réponse est OUI ou NON simplement sur la première lettre, soit O soit N.

INKEY$

Cette instruction permet la saisie d'un caractère au clavier, mais, à la différence de INPUT, elle n'attend pas qu'un caractère soit tapé, et n'envoit pas d'écho sur l'écran. Il est donc nécessaire de provoquer un test et une boucle pour savoir si une touche a été enfoncée.

Exemple :

```
10 PRINT"Taper un nombre entre 1 et 9"
20 A$="":'          CHAINE VIDE
30 A$=INKEY$:IFA$=""GOTO30
40 X%=VAL(A$)
50 PRINTX%
```

- Ligne 20 : on définit une chaîne vide A$.
- Ligne 30 : on affecte à A$ la touche enfoncée et on teste : si A$ est toujours vide on retourne en 30.
- Ligne 40 : X$ prend la valeur numérique entière du caractère A$.

Toutes les instructions du type INPUT, ainsi que INKEY$, attendent une réponse de l'utilisateur.

Nous allons étudier maintenant un ensemble d'instructions qui permettent au programmeur de faire lire par l'unité centrale des informations contenues dans le programme.

READ VAR, VAR...

Cette instruction permet d'affecter à la première variable la première information placée dans le programme derrière l'instruction DATA, puis d'affecter à la deuxième variable la deuxième information, etc.

Les variables sont de types quelconques, mais il doit y avoir correspondance entre le type d'une variable et son information.

DATA INF, INF,...

Cette instruction placée n'importe où dans le programme signale que ce qui la suit est l'ensemble des informations qui seront lues à la suite d'un READ.

Exemple :

```
10 CLS:PRINT"Voulez-vous connaitre les h
oraires ?"
20 X$=INPUT$(1)
30 IFX$="N"THENEND
40 READD$,H$,D1$,H1$,D2$,H2$,D3$,H3$
50 PRINTD$,H$
60 PRINTD1$,H1$
70 PRINTD2$,H2$
80 PRINTD3$,H3$
90 END
100 DATA"Reseau Est","10 Octobre 1988",N
ANCY,20H53,STRASBOURG,20H55,BESANCON,20H
59
```

On remarque qu'une chaîne contenant des blancs doit être placée entre guillemets.

RESTORE N

Cette instruction pointe le READ qui suivra vers les DATA situées ligne N.

Si N est omis, RESTORE pointe vers le premier DATA rencontré dans le programme pour, éventuellement, permettre une nouvelle lecture des informations.

Exemple :

```
50 PRINT"Voulez-vous connaitre le mot de
 passe ?"
60 X$=INPUT$(1)
70 IFX$="N"GOTO100
80 READMOT$,M
90 PRINTMOT$,M
100 DATA TROMBONE,74
110 '....
120 '...
200 RESTORE
210 PRINT"Voulez-vous connaitre le mot d
e passe ?"
220 X$=INPUT$(1)
230 IFX$="N"GOTO250
240 READMOT$,M
250 '...
```

Exemple :

```
10 PRINT"Voulez-vous connaitre les multi
ples de 2,3,ou 5 ?"
20 DATA"Multiples de 2",2,4,6,8,10,12,14
,16
30 DATA"Multiples de 3",3,6,9,12,15,18,2
1,24
40 X$=INPUT$(1)
50 DATA"Multiples de 5",5,10,15,20,25,30
,35,40
60 X=VAL(X$)
```

```
70 IFX=2THENRESTORE80ELSE IFX=3THENRESTO
RE90ELSERESTORE50
80 READT$,M1,M2,M3,M4,M5,M6,M7,M8
90 PRINTT$
100 PRINTM1,M2,M3,M4,M5,M6,M7,M8
110 GOTO10
```

Comme on a pu le constater dans les exemples précédents, l'instruction PRINT permet l'affichage d'un texte sur l'écran. Observons donc cette instruction d'un peu plus près.

PRINT TEXTE ; TEXTE

Cette instruction permet l'affichage sur l'écran des expressions qui la suivent.

Si les expressions sont séparées par un point-virgule, elles sont écrites l'une à la suite de l'autre sans espace supplémentaire.

Attention : n'oubliez pas que toutes les valeurs numériques se terminent par un espace, et commencent par un espace lorsqu'elles sont positives (sinon par –).

Chaque ligne est divisée en trois zones de 13 caractères :

Zone 1, de 0 à 12
Zone 2, de 13 à 25
Zone 3, de 26 à 38

Si les expressions sont séparées par une virgule, elles s'inscrivent sur l'écran à partir de chaque début de zone.

Si la dernière expression n'est suivie ni de la virgule ni du point-virgule, le PRINT suivant provoquera un retour à la ligne suivante colonne 0, sinon on reste sur la ligne en cours.

Si une expression chaîne a plus de 40 caractères, le saut à la ligne est automatique.

Attention : une valeur numérique ne peut s'inscrire sur deux lignes.

SPC(N)

Cette instruction placée derrière un PRINT introduit *N espaces.*

TAB(N)

Cette instruction placée derrière un PRINT ou un séparateur virgule ou point-virgule positionne le curseur dans la colonne de numéro N.

Si, lorsqu'il rencontre cette instruction, le curseur se trouve dans une colonne de numéro supérieur à N, alors il effectue un saut à la ligne avant de tabuler colonne N.

Exemple :

```
10 PRINT"BONJOUR",TAB(10);"MADAME"
```

Ecrira : BONJOUR ← ligne L

MADAME ← ligne L + 1

↑ Col Ø

↑ Col 1Ø

En effet, BONJOUR s'affiche colonne Ø. Puis la virgule positionne le curseur colonne 13, donc TAB (1Ø) se fait à la ligne suivante.

PRINT USING IMA ; VAR

Cette instruction permet l'affichage de la variable VAR suivant une IMAGE définie préalablement.

– *IMAGE d'une chaîne :* A$ = « % % »

n espaces

Toute chaîne qui utilisera A$ comme image sera affichée sur une longueur de n+2 caractères, n étant le nombre d'espaces existant entre les délimitateurs %. Si la longueur de la chaîne est supérieure à n+2, seuls les n+2 premiers caractères sont affichés.

Si la chaîne est plus courte que l'image, elle est complétée par des espaces à droite.

Deux autres possibilités sont offertes pour l'image d'une chaîne :

A$ = « ! » n'affichera que le premier caractère.

A$ = « & » affichera exactement la chaîne quelle que soit sa longueur.

Exemple :

```
10 A$="%    %":'          4 ESPACES
20 PRINTUSINGA$;"BONJOUR MONSIEUR"
30 PRINTUSING"!";"BONJOUR"
40 PRINTUSING"&";"MONSIEUR"
50 PRINTUSING"&";"BONJOUR","MONSIEUR"
```

Ce programme affichera : BOUJOU
B
MONSIEUR
BONJOURMONSIEUR

– *IMAGE d'une valeur numérique* : A$ = « # # # # »

Toute variable numérique utilisant l'image A$ sera arrondie à l'entier le plus proche, puis affichée avec cadrage à droite.

Si la variable est plus courte que l'image, on complète avec des blancs à gauche.

Si la variable est plus longue, elle est écrite dans sa totalité et précédée du symbole %.

On peut insérer dans l'image un point décimal pour séparer partie entière et partie décimale.

Exemple : A$ = « # # # · # # ».

Les règles précédentes restent valables. On peut faire apparaître le signe + en début de zone ou le signe – en fin de zone si la valeur est négative, en les faisant apparaître dans l'image.

Exemples :

La valeur 120.18 s'écrira 120.2 si A$ = « # # # · # »
ou + 120.2 si A$ = « + # # # · # »

La valeur – 12.41 s'écrira – 12.41 si A$ = « # # # · # # »
ou 12.41– si A$ = « # # # · # # – »

Si une variable est plus courte que son image, on peut la compléter avec le symbole ★ en faisant précéder l'image d'une double étoile (ex. : A$ = « ★ ★ # # # · # # »), ou simplement par un $ en faisant précéder son image d'un double $.

Exemples :

La valeur 120.18 s'écrira
★ ★ 120.18 si A$ = « ★ ★ # # # # # · # # »
ou $ 120.18 si A$ = « $ $ # # # # # · # # »
espace

L'affichage peut se faire avec la forme exponentielle si l'image est terminée par quatre flèches ↑ ↑ ↑ ↑.

4. APPLICATION

On résoudra le problème posé dans l'exemple d'introduction grâce au programme ci-dessous :

Exemples :

```
10 CLS:A$="":B$="%          %"
20 A$=INKEY$:IFA$=""GOTO20
30 A=VAL(A$)
40 IFA=1THENRESTORE100ELSEIFA=2THENRESTO
RE200ELSEIFA=3THENRESTORE300
50 READNL
60 FORN=1TONL
70 READNOM$,TIT$
80 PRINTUSINGB$;NOM$,TIT$
90 NEXTN:END
100 DATA2,BROWN,"MISS TRANSE",DEMOUZON,B
UNGALOW
200 DATA1,ANDREVON,NEUTRON
300 DATA1,DECAUX,....
```

CHAPITRE V **La gestion de l'écran semi-graphique**

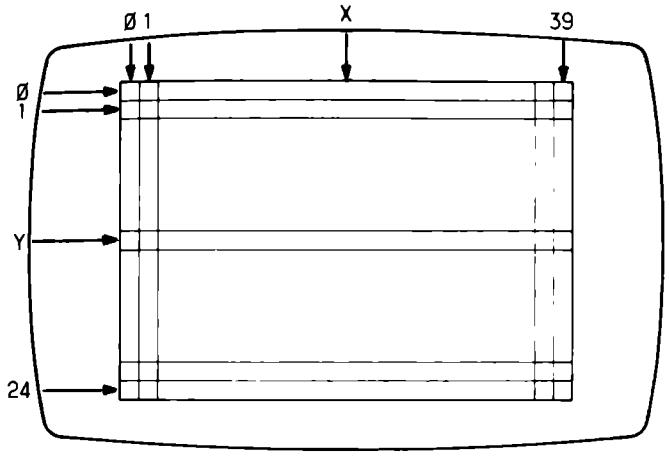
En mode semi-graphique, l'écran peut être défini comme un ensemble de 1 000 cases réparties en 25 lignes numérotées de Ø à 24 et de 40 colonnes numérotées de Ø à 39.

Une case sera donc définie par deux numéros :

- premièrement, son numéro de colonne X
- deuxièmement, son numéro de ligne Y

Par exemple, la case (1), la plus en haut et à gauche, est la case Ø,Ø. Celle qui est en bas, à droite, est la case 39,24.

(1) Chaque case ne peut contenir qu'un et un seul caractère semi-graphique, lettre, chiffre, symbole, etc.



ECRAN

On remarquera qu'en fait la case Ø,Ø n'est pas en haut et à gauche de l'écran, mais en haut et à gauche de la zone écran réservée à l'ordinateur.

Autour de cette zone réservée se trouve ce qu'on appellera *le CADRE*, dans lequel on ne peut en aucun cas écrire ou faire un dessin. Le cadre nous est inaccessible, mais on peut en choisir la couleur.

1. EXEMPLE INTRODUCTIF

Gérer l'écran, c'est, en fait, d'abord définir :

- la couleur du cadre,
- la couleur de fond d'une case,
- la couleur d'un caractère dans une case,
- la hauteur et la largeur des caractères,
- éventuellement, se réserver une zone d'écriture et masquer des caractères.

Puis, ensuite, il faut positionner un caractère dans l'écran pour l'écrire ou, au contraire lire un caractère dans l'écran pour relever sa position.

Prenons l'exemple du problème suivant :

1. Sur un écran totalement noir, je veux faire apparaître simultanément :

- un titre « BIBLIOTHEQUE » en caractères rouges de double largeur et double hauteur, en haut et au milieu de l'écran ;
- des chapitres numérotés de 1 à 6. Le numéro, vert, devra toujours être dans la deuxième colonne et apparaître sur fond jaune. Les sous-titres correspondants seront en jaune sur fond noir, de taille normale et commenceront à la sixième colonne ;
- le curseur sera localisé à la case Ø,Ø.

2. L'utilisateur pourra déplacer le curseur à l'aide des touches spécialisées et l'amener ainsi devant le chapitre qui l'intéresse, n'importe où sur la ligne correspondante.

- Une action sur ENTREE devra alors entraîner la localisation du curseur, pour en déduire le chapitre choisi.

Le problème est posé, examinons les outils dont nous disposons.

2. LES INSTRUCTIONS DE DEFINITION

ATTRB L, H, M

Cette instruction définit les attributs des caractères :

L = Ø ⟹ largeur standard
L = 1 ⟹ double largeur

H = Ø ⟹ hauteur standard
H = 1 ⟹ double hauteur

M = Ø ⟹ caractère masqué
M = 1 ⟹ caractère démasqué

Exemples :

```
10 ATTRB0,0
20 PRINT"Bibliotheque"
10 ATTRB1,0
20 PRINT"Bibliotheque"
```

```
10 ATTRB0,1
20 PRINT"Bibliotheque"

10 ATTRB1,1
20 PRINT"Bibliotheque"
```

Après chacun de ces exemples, le lancement par RUN fait apparaître le mot dans le caractère choisi.

Le masquage permet l'écriture sur l'écran, tout caractère écrit dans ce mode étant recouvert d'un masque noir.

Lorsque l'écran est composé, on peut démasquer l'ensemble avec l'instruction UNMASK.

UNMASK

Mais tous les caractères sont alors de la même couleur. Comparer, par exemple, les deux programmes ci-dessous :

```
10 CLS
20 SCREEN6,0,0
30 FORN=1TO1000
40   PRINT"A";
50   NEXT
```

Dans ce programme on voit les « A » remplir l'écran l'un après l'autre.

```
10 CLS
20 SCREEN6,0,0:ATTRB,,1
30 FORN=1TO1000
40   PRINT"A";
50   NEXT
60 UNMASK
```

Dans ce programme l'écran est soudain remplit de « A ».

CLS

Cette instruction efface la fenêtre définie par l'instruction CONSOLE et ramène le curseur en haut et à gauche de cette fenêtre.

Si l'instruction CONSOLE n'a pas été utilisée, CLS efface l'écran et renvoie le curseur à la case 0,0.

CLS est équivalent à l'appui sur la touche RAZ .

COLOR C, F, V

Cette instruction fixe la couleur :

C = du caractère (de Ø à 7).
F = du fond de la case où est inscrit ce caractère (de Ø à 7).

Le chiffre V (Ø ou 1) permet d'inverser la vidéo : V = Ø vidéo normale, V = 1 vidéo inversée.

Exemple :

```
10 COLOR4,6,0
20 PRINT"A"
```

Ecrira → A en bleu foncé sur fond bleu clair.

```
10 COLOR4,6,1
20 PRINT"A"
```


A en bleu clair sur fond bleu foncé.

Cette instruction n'affecte que la couleur d'une case de l'écran sans modifier les couleurs du reste de l'écran.

CONSOLE H, B, S, V

Cette instruction permet de redéfinir la fenêtre écran dans laquelle on souhaite travailler, ainsi que la vitesse de « scrolling » et son type (couleur ou non).

- H représente le numéro de la ligne haute de la fenêtre.
- B représente le numéro de la ligne basse de la fenêtre.

Ainsi l'instruction CONSOLE Ø,24 nous permet de travailler sur tout l'écran, alors que CONSOLE 1Ø,2Ø, ne nous donne que 11 lignes accessibles de la ligne 1Ø à la ligne 2Ø incluses.

Dans ce dernier cas, l'instruction CLS n'efface que la fenêtre de 11 lignes ainsi définie et ramène le curseur au début de la ligne n° 1Ø.

Exemples : comparer ces deux programmes

```
10 CLS
20 CONSOLE0,24
30 PRINT"A"
40 GOTO30
```

```
10 CLS
20 CONSOLE10,20
30 PRINT"A"
40 GOTO30
```

- S définit le type de scrolling
  S = Ø ⇒ scrolling en couleurs, S = 1 ⇒ scrolling sans couleurs.

Exemple : comparer ces deux programmes.

```
10 CLS
20 SCREEN4,6,6
30 COLOR1,4
40 CONSOLE,,0
50 FORN=1TO1000:PRINT"A";:NEXT
```

Dans ce premier programme, ligne 20, on définit l'écran (cadre et fond bleu ciel, caractères bleus foncés).

Puis, ligne 30, on définit un affichage de caractères rouges sur fond bleu ciel.

Ligne 40, la console ouvre par défaut la fenêtre maximum, c'est-à-dire de la ligne : à la ligne 24, avec scrolling en couleurs.

La ligne 50 remplit l'écran de A rouges sur fond bleu foncé. Le cadre reste bleu ciel.

Si l'on modifie la ligne 40 de la façon suivante :

```
40 CONSOLE,,1
```

Cela annule l'effet de l'instruction COLOR de la ligne 30. Mais cette instruction a été lue et enregistrée (mémorisée) par l'ordinateur, et, si après avoir commandé (en mode direct) RUN, vous commandez CONSOLE, 0, le TO 7 vous répond OK en caractères rouges sur fond bleu.

Si vous faites le programme suivant :

```
10 SCREEN4,6,6
20 COLOR1,4
30 CONSOLE10,20,0
40 CLS
50 FORN=1TO1000:PRINT"A";:NEXT
```

Vous ne remplissez de A rouges sur fond bleu ciel que la fenêtre 10,20.

Si vous modifiez la ligne 30 comme ci-dessous :

```
30 CONSOLE,,1
```

Vous ne modifiez pas la fenêtre qui est inchangée par défaut des constantes H et B, mais vous écrivez suivant la définition d'écran.

– V définit la vitesse du scrolling.
  V = Ø vitesse normale
  V = 1 vitesse lente
  V = 2 vitesse mode page.

Faites les trois essais dans le programme précédent en ajoutant successivement les trois valeurs de V.

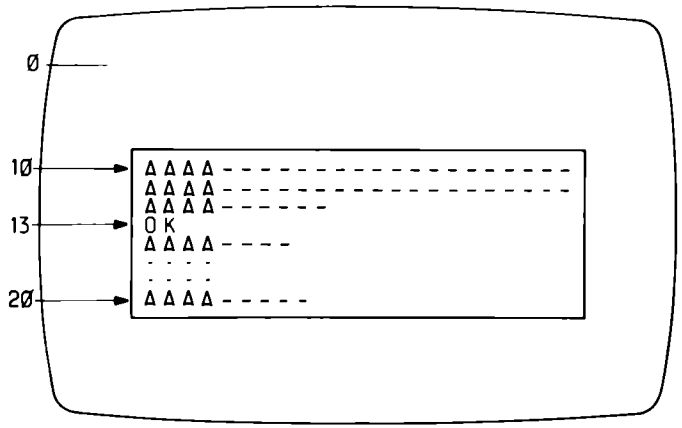
```
30 CONSOLE,,1,0
```
puis

```
30 CONSOLE,,1,1
```
et enfin

```
30 CONSOLE,,1,2
```

Dans ce dernier cas, lorsque la fenêtre est remplie, le curseur remonte à la case départ, c'est-à-dire ligne Ø colonne Ø, et continue son impression des A. Mais comme la fenêtre ligne Ø, ligne 9 n'est pas ouverte, rien n'apparaît sur l'écran.

Or le programme est écrit pour remplir l'écran de 1ØØØ « A », soit 24 lignes. Seules 11 lignes sont remplies quand le curseur remonte case Ø,Ø. Il en reste donc encore 13, dont 1Ø (lignes Ø à 9) restent invisibles, les trois derniers réapparaissent dans la fenêtre. Puis le TO7 précise OK qui apparaît donc ligne 13.



SCREEN C, F, K, V

Définit la vidéo écran :

- C définit la couleur des caractères (de Ø à 7)
- F définit la couleur du fond de la fenêtre (de Ø à 7)
- K définit la couleur du cadre (de Ø à 7)
- V définit la vidéo normale Ø ou inverse 1.

Voir les exemples précédents.

3. LES INSTRUCTIONS DE POSITIONNEMENT

LOCATE C, L, Q

Cette instruction place la curseur colonne C, ligne L.

Q indique si le curseur doit être visible (Q = 1) ou non (Q = Ø).

Par défaut le curseur est visible.

Attention : les valeurs de C et de L doivent toujours être spécifiées. Elles ne peuvent être prises par défaut.

Comparez les deux programmes ci-dessous :

```
10 CLS
20 SCREEN4,6,6,1
30 LOCATE11,11
40 ATTRB1,1
50 PRINT"BONJOUR"
60 LOCATE0,22
70 ATTRB0,0
```

Dans ce cas, le curseur est visible sous l'inscription OK.

Puis remplaçons la ligne 60 :

```
10 CLS
20 SCREEN4,6,6,1
30 LOCATE11,11
40 ATTRB1,1
50 PRINT"BONJOUR"
60 LOCATE0,22,0
70 ATTRB0,0
```

Dans ce cas, le curseur est invisible sous l'inscription OK.

PSET (C, L) « K », Q

Cette instruction écrit colonne C, ligne L, le caractère « K » de couleur Q.

Exemple : PSET (20, 10) « $ », 1 écrit à la 20e colonne et 10e ligne un $ rouge

A$ = « Ø »

PSET (20, 10) A$, Ø remplace le $ par un Ø noir.

Le caractère « K » peut également être un GR$.

LINE (C_1, L_1) – (C_2, L_2) « K », Q

Cette instruction trace une ligne du caractère K, de la case située colonne C_1, ligne L_1 à la case située colonne C_2, ligne L_2, dans la couleur Q.

Comme pour PSET, le caractère peut être un A$ ou GR$ ou CHR$.

Exemple :

```
10 CLS
20 SCREEN7,0,0
30 LINE(0,0)-(38,0)"-"
```

Ce programme trace une ligne pointillée en haut de l'écran en blanc sur fond noir.

POS

Cette instruction retourne le numéro de colonne du curseur.

Exemple :

```
10 FORL=0TO24
20 PSET(C,L)"-"
30 NEXT
40 C=POS
50 GOTO10
```

Lorsque vous faites RUN, toutes les variables sont réinitialisées donc C = L = Ø.

De la ligne 10 à 30, on remplit la colonne C = Ø de « – ». Une fois le dernier « – » écrit sur l'écran, le curseur se positionne à la case suivante, c'est-à-dire *colonne 1*, ligne 24.

Donc, en ligne 40, C prend la *valeur 1*, etc.

L'écran se remplit donc de « – » colonne après colonne et de haut en bas.

CSRLIN

Cette instruction retourne le numéro de la ligne du curseur. N'oubliez pas qu'il y a 25 lignes numérotées de Ø à 24. Avec l'instruction précédente, on peut connaître la position exacte du curseur sur l'écran.

On peut donc reprendre l'exemple précédent en remplissant l'écran ligne par ligne.

Exemple :

```
10 FORC=0TO39
20 PSET(C,L)"?",1
30 NEXT
40 L=CSRLIN
50 GOTO10
```

SCREEN (C,L)

Cette instruction retourne le code ASCII (voir annexe 7) du caractère situé colonne C, ligne L. Si aucun caractère connu ne se trouve en C,L, alors l'instruction retourne Ø (c'est le cas des caractères GR$ définis par l'utilisateur).

Exemple :

```
10 CLS
20 SCREEN0,7,7
30 PRINT"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
40 FORC=0TO25
50    X=SCREEN(C,0)
60    PRINTX;
70 NEXT
80 END
```

4. APPLICATION

Le problème posé en début de chapitre sera traité dans l'ordre logique suivant :

1. Effacer l'écran/définir l'écran caractères rouge sur fond noir, cadre noir/caractère double largeur, double hauteur.
2. Se placer ligne 1, colonne 13 et écrire « BIBLIOTHEQUE ».

3. Placer le numéro 1 (vert/fond jaune), 4e ligne, 2e colonne, puis écrire « – Romans Policiers » (jaune/fond noir).
4. Sauter 3 lignes et refaire 3 avec le nº 2 et « Série Fiction », etc.
5. Placer le curseur en Ø,Ø et attendre la frappe sur [ENTREE].
6. Lire la position du curseur.
7. En déduire le numéro du chapitre.

PROGRAMME BIBLIOTHEQUE

```
10 CLS:SCREEN1,0,0:ATTRB1,1,0
20 LOCATE8,1,0:PRINT"BIBLIOTHEQUE":UNMAS
K:ATTRB0,0
30 LOCATE2,4:COLOR2,3:PRINT"1":COLOR3,0
:PRINT" - Romans Policiers"
40 LOCATE2,7:COLOR2,3:PRINT"2":COLOR3,0
:PRINT" - Serie fiction"
50 LOCATE2,10:COLOR2,3:PRINT"3":COLOR3,
0:PRINT" - Romans historiques"
60 LOCATE2,13:COLOR2,3:PRINT"4":COLOR3,
0:PRINT" - Romans d'aventures"
70 LOCATE2,16:COLOR2,3:PRINT"5":COLOR3,
0:PRINT" - Livres d'enfants"
80 LOCATE2,19:COLOR2,3:PRINT"6":COLOR3,
0:PRINT" - Bandes dessinees"
90 LOCATE0,0,1:INPUT"",A$
100 X=CSRLIN-1
110 NL=(X-4)/3+1
```

CHAPITRE VI **Les caractères utilisateurs**

Comme nous l'avons vu dans le paragraphe précédent, l'écran semi-graphique est constitué de 1 000 cases réparties en 40 colonnes et 25 lignes. Dans chaque case, on peut faire apparaître un caractère, lettre ou symbole alphanumérique, signe de ponctuation...

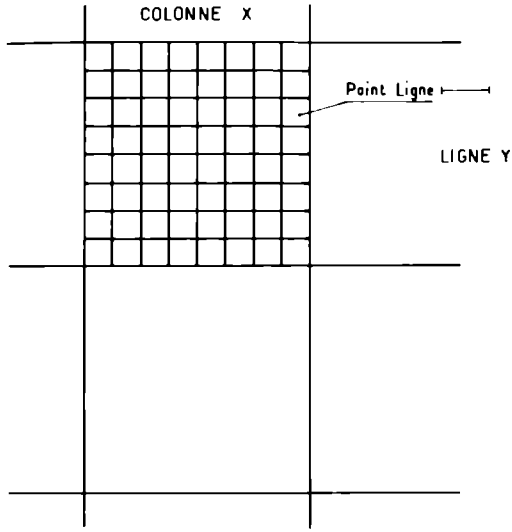


Fig. VI. 1.

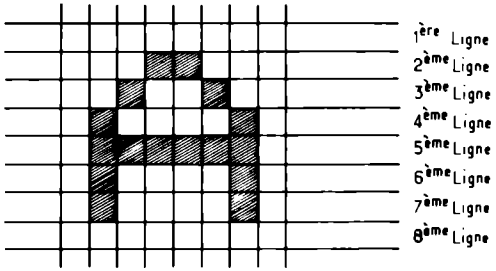
Comment est donc obtenu le dessin d'un caractère sur l'écran ? Eh bien, chaque case est divisée en un quadrillage de 8 × 8 (fig. VI-1), soit 64 points lumineux que l'on appellera points ligne (PL). Nous savons que chaque case est définie par une couleur de fond et une couleur de caractère, c'est-à-dire que chaque PL (1) peut être allumé dans la couleur de fond ou dans la couleur du caractère. Nous adopterons le système suivant :

- Lorsqu'un PL sera allumé dans la couleur du fond, nous dirons qu'il est à zéro (Ø).
- Lorsqu'un PL sera allumé dans la couleur du caractère, nous dirons qu'il est à un (**1**).

Comment est constituée la lettre A majuscule qui apparaît sur l'écran ? Comme on le constate sur la figure VI-2, certains PL sont à Ø et d'autres à **1**. Tout d'abord, un cadre de 1 PL de large est défini autour du caractère. Afin que les lettres qui se suivent à une case ne se touchent pas. La première ligne de 8 PL (que l'on appellera

(1) Les Anglo-Saxons disent « pixel ».

Fig. VI.2.



GPL : groupe de points ligne) ne contient que des PL à Ø, ce que l'on représente comme ceci :

première ligne = Ø Ø Ø Ø Ø Ø Ø Ø

La deuxième ligne voit 2 PL à 1 former la partie supérieure de la lettre A, donc :

deuxième ligne = Ø Ø Ø **1** **1** Ø Ø Ø

Finalement la lettre A est donc codée comme ceci :

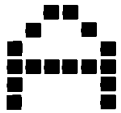
Ø Ø Ø Ø Ø Ø Ø Ø
Ø Ø Ø **1** **1** Ø Ø Ø
Ø Ø **1** Ø Ø **1** Ø Ø
Ø **1** Ø Ø Ø Ø **1** Ø
Ø **1** **1** **1** **1** **1** **1** Ø
Ø **1** Ø Ø Ø Ø **1** Ø
Ø **1** Ø Ø Ø Ø **1** Ø
Ø Ø Ø Ø Ø Ø Ø Ø

Or, dans le système de numération binaire (qui n'utilise que des Ø et des 1), on compte selon le code suivant :

Valeur décimale	Valeur binaire codée sur un octet (8 bits)
Ø	Ø Ø Ø Ø Ø Ø Ø Ø
1	Ø Ø Ø Ø Ø Ø Ø 1
2	Ø Ø Ø Ø Ø Ø 1 Ø
3	Ø Ø Ø Ø Ø Ø 1 1
4	Ø Ø Ø Ø Ø 1 Ø Ø
5	Ø Ø Ø Ø Ø 1 Ø 1
6	Ø Ø Ø Ø Ø 1 1 Ø
7	Ø Ø Ø Ø Ø 1 1 1
8	Ø Ø Ø Ø 1 Ø Ø Ø
etc.	

Donc un 1 dans la colonne de droite vaut 1, alors qu'un 1 dans la deuxième colonne vaut 2, puis 4 dans la troisième, 8 dans la quatrième, 16... 32... 64... 128 dans la colonne de gauche.

On peut donc représenter le caractère A par les valeurs décimales de chacun des GPL qui le composent, soit :



=

Ø	
24	⟶ 16 + 8
36	⟶ 32 + 4
66	etc.
126	
66	
66	
Ø	

Ce que l'on peut écrire :

caractère (A) = Ø, 24, 36, 66, 126, 66, 66, Ø

1. EXEMPLE INTRODUCTIF

Notre micro-ordinateur contient déjà un jeu de caractère figé sur lequel nous ne pouvons intervenir, mais son Basic va nous permettre de créer nos propres caractères, et ce, jusqu'à 128 caractères utilisateurs au maximum.

Supposons que nous ayons besoin de définir des lettres de l'alphabet grec, ou bien un message en chinois ou en arabe.

Par exemple :

La lettre q en arabe (qâf) s'écrit ق. Créons ce caractère.

Pour cela, dessinons-le dans un quadrillage 8 × 8 (fig. VI-3) notons **1** les PL constituant le caractère, et Ø ceux constituant le fond.

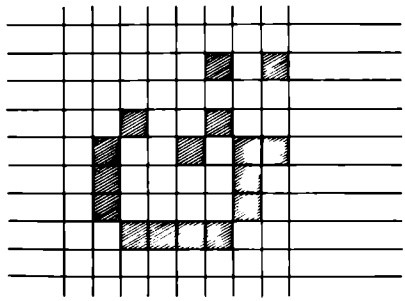


Fig. VI.3.

2. LES INSTRUCTIONS DE DEFINITION

DEFGR$(X)

Cette instruction permet de définir le caractère utilisateur GR$ de numéro X. On définit ce caractère en donnant les 8 valeurs décimales représentant le code binaire des GPL constituant le caractère.

On commence par la ligne du haut et on termine par celle du bas.

Exemple : je veux créer la lettre A majuscule qui s'appellera GR$ (Ø). J'écris donc : DEFGR$(Ø) = Ø, 24, 36, 66, 126, 66, 66, Ø

Exemple : la lettre arabe q sera mon caractère GR$ (1) si j'écris :

DEFGR$(1) = Ø, 5, Ø, 36, 75, 66, 66, 6Ø

CLEAR,,N

Cette instruction est nécessaire pour réserver de la place en mémoire afin d'y ranger les valeurs définissant chaque ligne de chaque caractère utilisateur.

La valeur N indique combien de caractères vont être créés par la suite. L'instruction CLEAR,,N doit donc précéder toute définition de GR$.

Le nombre d'octets réservés en mémoire sera de 8 × N. On peut créer au maximum 128 caractères, numérotés de Ø à 127. Une fois créé, un caractère se manipule comme une chaîne. On peut donc afficher son caractère en tapant PRINT GR$(X).

On peut accoler deux caractères par concaténation...

Exemple : A$ = GR$(Ø) + GR$(1)

A$ devient un nouveau caractère de 2 cases de longueur.

3. APPLICATIONS

– Rangement et pointage des GR$

Les valeurs décimales définissant les GR$ sont rangées en fin de mémoire utilisateur. Par exemple, dans sa version de base, le TO 7 offre à l'utilisateur 8 K de mémoire vive RAM de &H61ØØ à &H7FFF.

Si l'on définit un caractère GR$(Ø) = 1, 2, 3, 4, 5, 6, 7, 8, par exemple, on sait que 1 représente la valeur décimale des PL du haut du caractère et 8 la valeur décimale des PL du bas du caractère.

Lors de la définition, le TO 7 range ces valeurs dans l'ordre où il les trouve, au sommet – 1 de la mémoire, c'est-à-dire que que la valeur 1 est rangée en &H7FFE, 2 en &H7FFD, etc., jusqu'à 8, qui est rangée en &H7FF7.

Un planning de cette zone de RAM correspond alors au schéma ci-dessous :

Adresse RAM	&H7FF7	&H7FF8	&H7FF9	&H7FFA	&H7FFB	&H7FFC	&H7FFD	&H7FFE	&H7FFF
Contenu (décimal)	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Ø**

L'adresse de départ de la table des GR$ est pointée en RAM aux adresses &H6Ø2D-6Ø2E et &H612A-612B

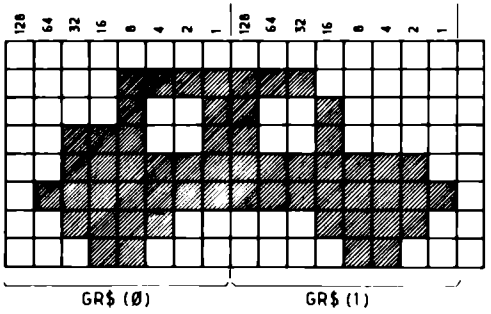
Dans l'exemple précédent, dès que je fais CLEAR 1, je trouve :

Adresse RAM	&H6Ø2D	&H6Ø2E						&H612A	&H612B
Contenu (hexadéc.)	**7F**	**F7**						**7F**	**F6**

Donc, dès que l'on déclare par CLEAR,,1 que l'on va créer un caractère utilisateur, le TO 7 pointe en &H6Ø2D.6Ø2E l'adresse de la RAM qui va contenir la valeur de la ligne du bas du caractère, ici &H7FF7, alors qu'en &H612A-612B on pointe la case libre immédiatement avant le premier GR$.

Si l'on définit N caractères, le pointeur &H6Ø2D-6Ø2E désigne le segment du bas du GR$(Ø) ; 8 cases plus loin on trouvera le segment du bas du GR$(1)... et finalement, en &H7FFE, on trouve la valeur du segment du haut du GR$(N–1).

– *Dessin et déplacement d'une automobile*



GR$(Ø) = { Ø, 15, 9, 57, 63, 127, 6Ø, 24 }

GR$(1) = { Ø, 224, 144, 144, 254, 255, 3Ø, 12 }

```
10 CLS:SCREEN7,0,0:ATTRB1,1
20 CLEAR,,2
30 DEFGR$(0)=0,15,9,57,63,127,60,24
40 DEFGR$(1)=0,224,144,144,254,255,30,12
50 V$=" "+GR$(0)+GR$(1)
60 FORX=0 TO 32
70   LOCATEX,12:PRINTV$;
80   NEXT
90 CLS:GOTO60
```

– *Sauvegarde des GR$ sous forme de fichier*

On peut sauvegarder les GR$ sur la cassette, sous forme de fichier binaire (voir plus loin ce chapitre), à l'aide du programme ci-dessous :

```
10 CLEAR,,2
20 DEFGR$(0)=0,15,9,57,63,127,60,24
30 DEFGR$(1)=0,224,144,144,254,255,30,12
40 AH=&H8FFF:' Ou bien &H7FFF en version
 8k.
50 AB=AH-2*8
60 SAVEM"CAS8:AUTO",AB,AH,AB
```

On peut ensuite le recharger à sa place à l'aide des lignes de programme ci-dessous :

```
10 CLEAR,,2
20 LOADM"CAS0:AUTO"
```

Si le nombre des GR$ est X. on écrit évidemment.

```
50 AB=AH-X*8
```

Remarque : Lorsque l'on a défini un certain nombre de GR$, on peut par la suite, les utiliser comme des CHR$

Exemple :
PRINT CHR$ (128 + X) est équivalent à
PRINT GR$(X)

Donc, les CHR$(128) → CHR$(255) représentent les
GR$(Ø) → GR$(127).

CHAPITRE VII **Branchements, itérations et sous-programmes**

1. EXEMPLE INTRODUCTIF

Pour calculer le montant de l'impôt sur le revenu d'un salarié, on procède de la façon suivante :

1. Calcul du revenu imposable (RI) égal à 80 % x 90 % du revenu net + pensions et rentes – déductions.
2. Calcul du quotient familial (QF) égal à RI divisé par le nombre de parts NP.
3. Recherche de la tranche et application de la formule correspondante.

On se propose donc de réaliser un programme de calcul de l'impôt simplifié (pas de corrections éventuelles) qui permette de modifier facilement d'une année sur l'autre les minima et maxima du

quotient familial de chaque tranche ainsi que les taux et coefficients de chacune (fig. VII-1).

QF minimum	QF maximum	Taux	Coefficient
0	12 620	0	0
12 620	13 190	0,05	631
13 190	15 640	0,10	1 290,5
15 640	24 740	0,15	2 072,5
24 740	31 810	0,20	3 309,5
31 810	39 970	0,25	4 900
39 970	48 360	0,30	6 898,5
48 360	55 790	0,35	9 316,5
55 790	92 970	0,40	12 106
92 970	127 860	0,45	16 754,5
127 860	151 250	0,50	23 147,5
151 250	172 040	0,55	30 710
172 040	195 000	0,60	39 312
195 000		0,65	49 062

Fig. VII-1. – Tableau des divers coefficients pour l'année 1982.
Impôt = (RI x Taux) – (Coeff. x NP) si $QF_{min} \leqslant QF \leqslant QF_{max}$.

Le programme fera appel à un sous-programme pour la lecture de ces coefficients, taux..., et à un autre sous-programme pour le calcul de l'impôt.

On peut donc décomposer le programme suivant l'organigramme de la page suivante (fig. VII-2).

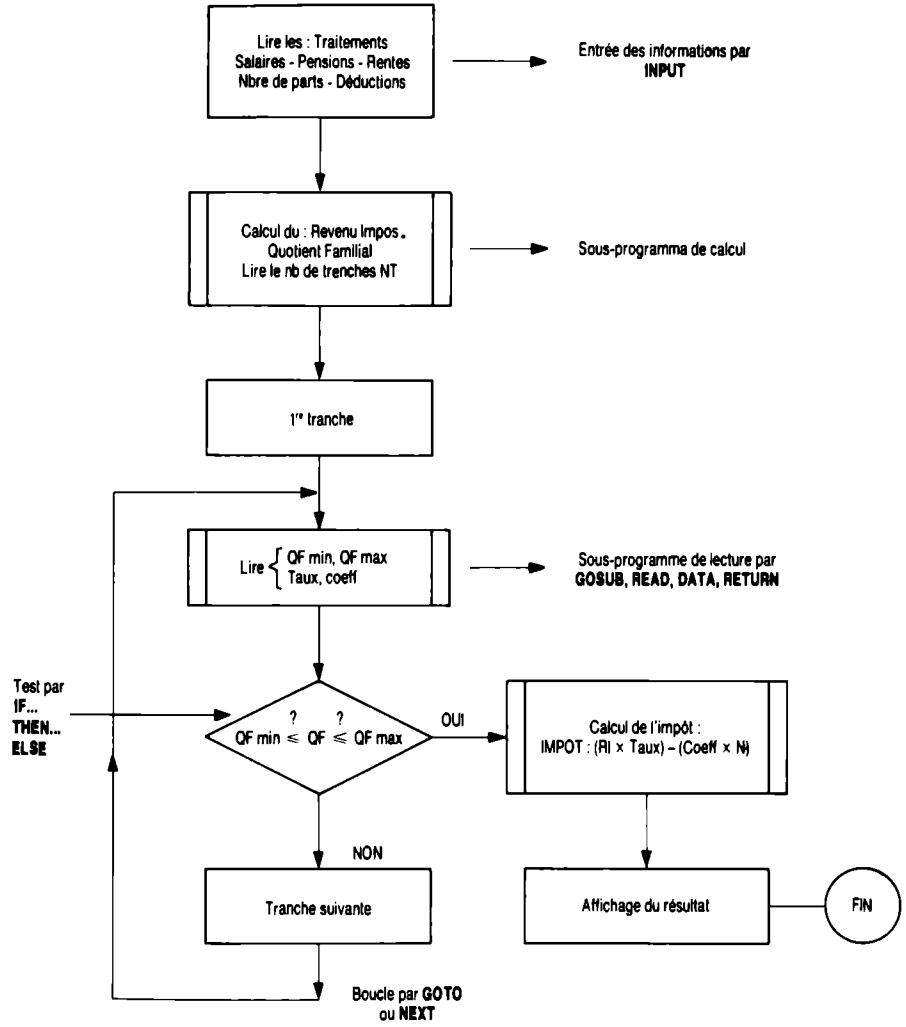


Fig. VII.2. – Organigramme.

2. BRANCHEMENTS INCONDITIONNELS

GOTO NL

Cette instruction force l'unité centrale à sauter à la ligne indiquée derrière GOTO.

ON VAR GOTO NL1, NL2,...

Suivant la valeur de la variable VAR, cette instruction force à la ligne NL1 si VAR = 1, NL2 si VAR = 2, etc.

Exemple :

```
10 PRINT"Quel chapitre choisissez-vous "
;
20 INPUTX
30 ON X GOTO100,200,300
```

3. BRANCHEMENTS CONDITIONNELS

IF... THEN... ELSE

Cette instruction permet de réaliser un test dont la réponse sera OUI ou NON. Si le test est vrai (réponse OUI), alors seules les instructions qui suivront THEN seront prises en compte.

Si le test est faux (réponse NON), alors seules les instructions suivant ELSE seront prises en compte.

Exemple :

```
10 INPUT"Quel est votre age :";AGE
20 IF AGE>10THEN PRINT"Vous etes trop ag
e" ELSE GOTO100
```

Les tests peuvent être multiples :

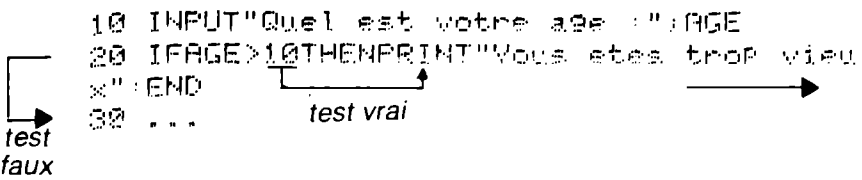
Exemple :

```
10 INPUT"Quel est votre age :";AGE
20 IF AGE>10THENGOTO50ELSE IF AGE<5 THEN
PRINT"Vous etes trop jeune"ELSE GOTO100
```

L'instruction THEN GOTO peut être remplacée par GOTO seule.

On peut omettre l'instruction ELSE. Dans ce cas, si le test est faux, le programme passe à la ligne suivante.

Exemple :



Dans l'exemple ci-dessus, on remarque que plusieurs instructions peuvent être lues en séquence, que le test soit vrai ou faux, à condition de les séparer par « : ».

Exemple :

```
10 INPUT"Quel est votre age :";AGE
20 IFAGE>10THENPRINT"Vous etes trop vieu
x":END ELSEIF AGE <5 THENPRINT"Vous etes
 trop jeune":END ELSE GOTO100
30 ...
```

Opérateurs de test AND, OR, NOT, XOR, >, <, =

Le test suivant « Si A est plus grand que 2 et si B est plus petit ou égal à 3, alors aller ligne 100, sinon aller ligne 200 » peut s'écrire :

```
20 IF A>2 AND B<=3 GOTO100 ELSE GOTO200
```

L'opérateur OR permet le test OU INCLUSIF.
L'opérateur NOT permet l'inversion du test.
L'opérateur XOR permet le test OU EXCLUSIF.
L'opérateur <> permet le test « différent de ».

FOR...TO...STEP.../NEXT

Cet ensemble d'instructions permet la répétition de toute la partie du programme comprise entre l'ouverture de la boucle par FOR... TO... STEP et la fermeture par NEXT.

A l'ouverture de la boucle, on indique qu'une variable VAR doit être incrémentée d'une valeur initiale I jusqu'à une valeur finale F par pas de P.

Exemple : FOR X = 1 TO 10 STEP 1

Quand le pas est de 1, on peut omettre cette partie.

Exemple : FOR Z = 2 TO 16 STEP 2.

Lorsque le programme arrive à l'ouverture de la boucle, il affecte à la variable la valeur initiale, puis traite les lignes suivantes jusqu'à ce qu'il rencontre l'instruction NEXT.

A cet instant, il incrémente la variable de la valeur du pas et traite les lignes suivantes avec cette nouvelle valeur, etc., jusqu'à ce que la variable atteigne la valeur finale. Les instructions sont lues une fois de plus, la variable est encore une fois incrémentée d'un pas, mais le programme ne revient plus en arrière et, au contraire, continue en séquence après l'instruction NEXT.

Exemple : on veut afficher la table de multiplication par 3, de 3 x 1 à 3 x 10.

```
10 CLS:PRINT"Table de Multiplication par
  3"
20 FORN=1TO10
30 PRINT"3*";N;"=";3*N
40 NEXT
50 END
```

On peut imbriquer plusieurs boucles ayant des variables différentes à condition d'indiquer (fig. VII-3) le nom de la variable derrière l'instruction NEXT.

Tous les schémas de programmes ci-dessus correspondent à des imbrications de boucles correctes. Par contre, le schéma de programme de la figure VII-4 n'est pas correct.

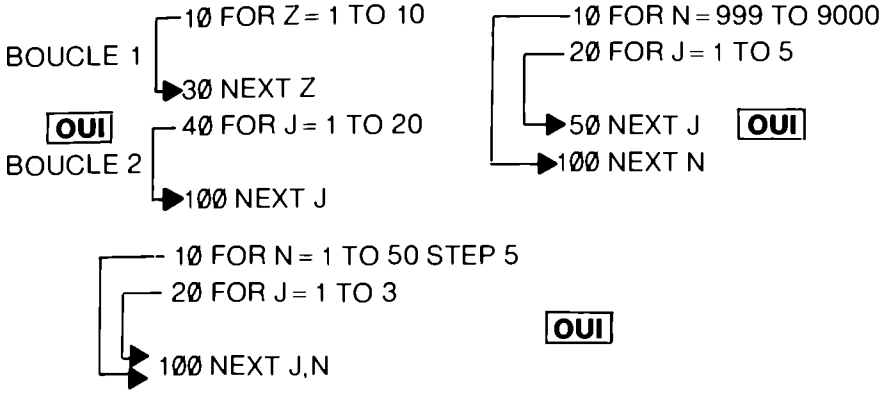


Fig. VII-3.

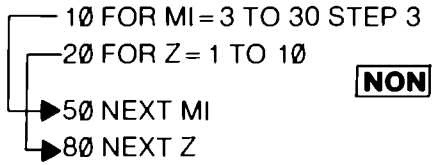
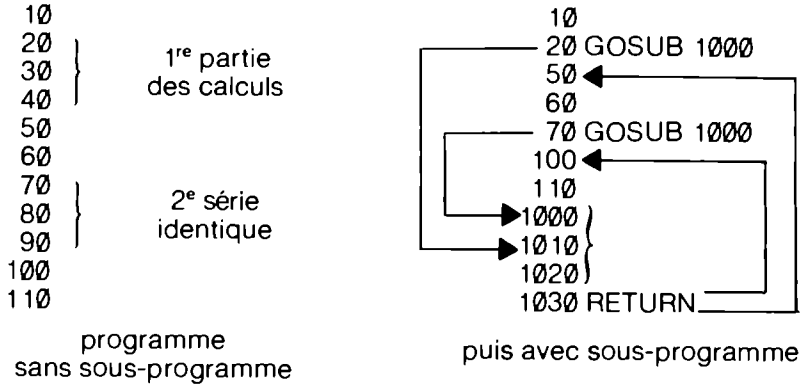


Fig. VII-4

5. BRANCHEMENTS A DES SOUS-PROGRAMMES

Lorsque, dans un programme, un calcul ou une série de calculs ou d'opérations sont effectués plusieurs fois à des lignes différentes, il est intéressant de faire de cette série de calculs ou d'opérations un sous-programme, que l'on appelle à partir du programme principal par l'instruction GOSUB suivie du numéro de la première ligne du sous-programme, et qui se termine par RETURN pour reprendre en séquence le programme principal.

Exemple :



GOSUB NL

Cette instruction dirige vers le sous-programme commençant à la ligne NL.

ON Var GOSUB NL1, NL2 :

Cette instruction agit comme ON GOTO, avec cette fois appel d'un sous-programme.

RETURN

Cette instruction termine nécessairement un sous-programme et permet le retour en séquence dans le programme principal.

D'un sous-programme on peut appeler un deuxième sous-programme, etc., sans limitation.

De même, d'un sous-programme on peut appeler ce sous-programme lui-même (récursivité).

Un sous-programme peut avoir plusieurs points d'entrée et plusieurs RETURN.

EXEC AD

Cette instruction permet d'aller réaliser un sous-programme écrit en langage 6809 et commençant à l'adresse AD.

Ce sous-programme doit commencer par sauvegarder la valeur du registre de pile S et du registre de page zéro DP (voir chapitre sur le 68Ø9) pour les restituer avant de revenir au programme Basic.

Ce sous-programme doit se terminer par l'instruction Assembleur RTS qui est l'équivalent de RETURN mais en langage machine (code &H39).

Le moniteur existant en mémoire morte contient déjà un certain nombre de sous-programmes que l'on peut utiliser (voir annexe 5).

Par exemple, en &HE82D se trouve le sous-programme appelé MENU$, qui est le point d'entrée à chaud dans le moniteur et qui renvoie la mire initiale.

Exemple : EXEC &HE82D [ENTREE].

En &HE82D, on trouve en fait les instructions suivantes :

&HE82D 7E EF 26, c'est-à-dire JMP $ EF26

Donc, en E82D se trouve une instruction de saut inconditionnel à l'adresse $EF26, qui est le point réel d'entrée à chaud du moniteur.

Cette instruction ne permet pas d'échanger de paramètre avec le sous-programme Assembleur. Dans ce cas, il faut utiliser les instructions suivantes.

DEF USR N° = AD

Cette instruction permet de définir les adresses de début de sous-programmes écrits en langage 68Ø9. Le numéro facultatif peut permettre la création de 1Ø sous-programmes numérotés de Ø à 9.

USR N° (Arg)

Cette instruction, à N° facultatif mais argument obligatoire, lance vers le sous-programme dont l'adresse a été définie précédemment.

Cette instruction permet l'échange d'un argument entre le sous-programme et le Basic.

Le type de l'argument est codé et le code rangé dans l'accumulateur A du 68Ø9 à l'appel comme au retour du sous-programme.

Le code est le suivant :

2 pour un entier
3 pour une chaîne
4 pour un réel
8 pour un réel double précision.

L'adresse du premier octet de l'argument est rangée dans l'accumulateur X (registre d'index 16 bits) à l'appel comme au retour.

CLEAR, AD

Cette instruction indique l'adresse maximum AD utilisable par le Basic.

Donc, à partir de AD+1, l'espace est réservé pour ranger un sous-programme en Assembleur. Cette instruction précède nécessairement DEFUSR et USR.

Exemple : on veut réaliser un sous-programme en Assembleur qui incrémente une variable Z définie et récupérée dans le Basic. Ce sous-programme doit être implanté à partir de &H7FØØ.

Lorsque l'on va passer la variable Z entière au sous-programme, le registre d'index X pointera vers le premier octet de la mémoire de stockage des variables numériques.

La variable étant entière, il faudra donc incrémenter le contenu de (X) + 3.

En Assembleur 68Ø9, on écrit : INC Ø3,X $6C $03
RTS ⟶ $39

On en déduit le programme complet.

```
10 CLEAR,&H7EFF
20 DEFINTZ:Z=2
30 DEFUSR1=&H7F00
40 FOR N=&H7F00 TO &H7F02
50 READX
60 POKEN,X
70 NEXT
80 Z=USR1(Z)
90 PRINTZ:GOTO80
100 DATA&H6C,&H03,&H39
```

```
10 CLS:SCREEN7,0,0
20 LOCATE3,2,0:ATTRB1,1:PRINT"CALCUL DE
L'IMPOT":ATTRB0,0
30 LOCATE0,6:INPUT"Total des salaires
",A
40 LOCATE0,8:INPUT"Frais reels si option
  : ",B
50 LOCATE0,10:INPUT"Deduction forfaitair
e sup.  : ",D
60 LOCATE0,12:INPUT"Pensions,retraites,.
..-10% : ",G
70 LOCATE0,14:INPUT"Charges a deduires
 ",J
80 LOCATE0,16:INPUT"Nb de Parts    ",N
90 GOSUB1000
100 FORX=1 TO NT
110 GOSUB2000
120 IFQF>QFMI AND QF<=QFMA THENGOSUB3000
:LOCATE3,2:ATTRB1,1:PRINT"IMPOT: F.";I1:
ATTRB0,0:END ELSE NEXTX
130 DATA14,0,12620,0,0,12620,13190,0.05,
631,13190,15640,0.1,1290.5,15640,24740,0
.15,2072.5,24740,31810,0.2,3309.5,31810,
39970,0.25,4900,39970,48360,0.3,6898.5,4
8360,55790,0.35,9316.5
140 DATA55790,92970,0.4,12106,92970,1278
60,0.45,16754.5,127860,151250,0.5,23147.
5,151250,172040,0.55,30710,172040,195000
,0.6,39312,195000,999999,0.65,49062
150 END
```

```
990 '
1000 '        CALCUL DU REVENU
1005 '        ================
1010 '
1020 IFB=0THENE=A*0.1+D ELSE E=B
1030 F=A-E
1040 H=0.2*(F+G)
1050 RB=F+G-H:LOCATE0,18:PRINT"Revenu br
ut  :";RB
1060 RI=RB-J:LOCATE0,20:PRINT"Revenu imp
osable  :";RI
1070 QF=RI/N:LOCATE0,22:PRINT"Quotient f
amilial  :";QF
1080 READ NT
1990 '
2000 '        LECTURE D'UNE TRANCHE
2005 '        =====================
2010 '
2020 READ QFMI,QFMA,TAUX,COEFF
2030 RETURN
2990 '
3000 '        CALCUL DE L'IMPOT
3005 '        =================
3010 '
3020 I1=(RI*TAUX)-(COEFF*N)
3030 RETURN
```

CHAPITRE VIII **Les variables numériques**

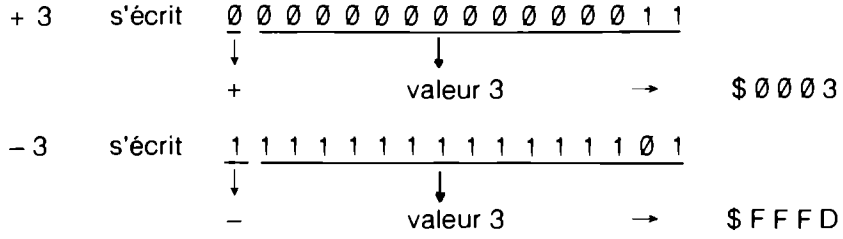
1. REPRESENTATION INTERNE

Comme nous l'avons vu au chapitre IV (Les bases du langage, page 35), les variables numériques peuvent être entières (%), simple précision (!) ou double précision(#).

Quel que soit le type de variable, sa valeur est rangée en mémoire selon une écriture binaire.

Les variables entières occupent chacune 2 octets, soit 16 bits par variable. Le bit de poids le plus fort est utilisé pour représenter le signe de la variable (Ø si +, 1 si –). Si la variable entière est positive, les 15 autres bits donnent sa valeur en mode binaire (maximum : 2^{15} = 32768, donc de Ø à 32767). Si la variable entière est négative, les 15 autres bits donnent la valeur en mode complément à deux (donc de – 1 à – 32768).

Exemple :



Les variables simple précision occupent 4 octets en mode virgule flottante. Ce mode virgule flottante permet une importante économie de place en mémoire, et fonctionne de la façon suivante :

Tout nombre est représenté par ses chiffres significatifs (la mantisse M) et un exposant E. La valeur du nombre N est alors :

$$N = (M + 0{,}5) * 2^{E-128}$$

L'exposant E est contenu en binaire dans le premier octet, les trois autres octets constituant la mantisse. Le bit de poids le plus fort de cette mantisse représente le signe du nombre.

Exemple :

N	81	Ø Ø	Ø Ø	Ø Ø

mantisse positive M = Ø

exposant E = $81 = 8 * 16 + 1 = 129

N = 0,5 * 2 = 1

N	82	Ø Ø	Ø Ø	Ø Ø

N = 0,5 * 4 = 2

N	83	C 4	Ø Ø	Ø Ø

mantisse : bit de poids fort à 1 → N négatif

M = (C-8) * 16^{-1} + 4 * 16^{-2} = 0,25 + 0,015625 = 0,265625

exposant : E = $83 = 8 * 16 + 3 = 131

N = – (0,265625 + 0,5)2^3 = – 6,125

Les variables double précision sont représentées également en mode virgule flottante mais occupent cette fois 8 octets en mémoire, dont 1 pour l'exposant et 7 pour la mantisse.

Les variables et leurs valeurs sont rangées à la suite du programme Basic. Chaque variable est précédée d'une valeur qui précise son type ; cette valeur est :

$3Ø pour les chaînes de caractères (3 octets)
$2Ø pour les variables entières (2 octets)
$4Ø pour les variables simple précision (4 octets)
$8Ø pour les variables double précision (8 octets)

Exemple : Dans un programme, on déclare à la ligne 1Ø les valeurs des variables A, B et C :

```
1Ø A = 1 : B# = 2 : C% = 3
```

On retrouve en fin de programme en mémoire :

```
ØØ ØØ ØØ | 4Ø | 41 81 ØØ ØØ ØØ | 8Ø | 42 82 ØØ ØØ ØØ ØØ ØØ ØØ ØØ
indique la fin du programme | A=Exp | mantisse | B=Exp | mantisse

2Ø | 43 ØØ Ø3 | ...
     C = valeur binaire
```

Une variable non déclarée (comme A dans l'exemple ci-dessus) est considérée comme simple précision et occupe donc 4 octets (au lieu des 2 nécessaires ici). On voit donc l'intérêt de spécifier le type de variable.

On peut également définir ce type pour plusieurs variables à l'aide des instructions ci-dessous :

DEFINT A

Cette instruction indique que toutes les variables non spécifiées commençant par la lettre A sont des variables entières.

Exemple :

A	est une variable entière
ALG	est une variable entière
A#	est une variable double précision
A et A#	sont considérées comme deux variables distinctes.

DEFSNG A-C

Cette instruction indique que toutes les variables non spécifiées commençant par les lettres A, B ou C sont des variables simple précision.

DEFDBL B-E, U

Cette instruction indique que toutes les variables non spécifiées commençant par les lettres B, C, D, E ou U sont du type double précision.

Il existe également, pour les chaînes de caractères, l'instruction DEFSTR.

2. LES OPERATEURS ARITHMETIQUES

Ces opérateurs sont, par ordre de priorité décroissante :

↑	Puissance	ex. : 2 ↑ 2
–	Moins (signe)	ex. : – 3
* et /	Multiplication et division	ex. : 4 * 3/5
@	Division entière	ex. : 5 @ 2 = 2
MOD	Reste d'une division	ex. : 5MOD2 = 1
+ et –	Addition et soustraction	ex. : 4 + 3 – 8

L'élévation à la puissance se fait en mode shifté (point jaune).

3. LES OPERATEURS RELATIONNELS

Ce sont les opérateurs classiques :

= Egalité
< Inférieur
<= Inférieur ou égal
> Supérieur
>= Supérieur ou égal
<> Différent de

Le résultat d'un test est une valeur binaire Ø ou – 1 suivant que le test est faux ou vrai. Cette valeur peut d'ailleurs être affectée à une variable et utilisée par la suite.

Exemple :

```
10 INPUTA,B,C,D:X=A+B<=C/D
20 IFX=0THENPRINT"A+B est Plus grand que
 C/D"
30 PRINT"X = ";X
```

4. LES FONCTIONS ARITHMETIQUES ET ALGEBRIQUES

ABS (Arg) Cette instruction calcule la valeur absolue de l'argument entre parenthèses et donne un résultat du même type que l'argument.

CDBL (Arg) Cette instruction convertit l'argument en type double précision.

CINT (Arg) Cette instruction convertit l'argument en type entier.

COS (Arg) Cette instruction calcule le cosinus de l'argument exprimé en radians.

CSNG (Arg) Cette instruction convertit un argument double précision en type simple précision.

EXP (Arg) Cette instruction calcule l'exponentielle de l'argument.

FIX (Arg) Cette instruction donne la partie entière signée de l'argument. Il ne s'agit pas d'une conversion, et le résultat est encore rangé en mémoire comme une valeur réelle.

INT (Arg) Cette instruction donne la valeur entière immédiatement inférieure ou égale à l'argument.

LOG (Arg) Cette instruction calcule le logarithme népérien de l'argument.

RND (Arg) Cette instruction permet d'obtenir une valeur pseudo-aléatoire, réelle, comprise entre Ø et 1 inclus. Si l'argument est négatif, plusieurs appels de cette instruction donnent la même valeur, sinon la valeur change à chaque fois.

SGN (Arg) Cette instruction donne le signe de l'argument.

SIN (Arg) Cette instruction calcule le sinus de l'argument (en rad.).

SQR (Arg) Cette instruction calcule la racine carrée de l'argument.

TAN (Arg) Cette instruction calcule la tangente de l'argument exprimée en radians.

5. APPLICATIONS

- ***Conversion d'un nombre entier inférieur à 256 en sa valeur binaire***

Ce nombre sera codé sur 8 bits que l'on appellera B(Ø), B(1),... B(7) en partant du poids faible jusqu'au poids fort. Donc :

$$N = B(7) * 2^7 + B(6) * 2^6 + \ldots + B(Ø) * 2^0$$

On en déduit donc que :

– B(7) est la partie entière de la division de N par 2^7
– B(6) est la partie entière de la division de N – B(7) * 2^7, par 2^6, etc.

La structure du programme sera donc la suivante :

1. Introduire la valeur de N.
2. Ouvrir une boucle de J = 7 à J = Ø
   – calculer B(J) = INT (N/(2 ↑ J))
   – poser N = N – B(J) * 2^J.
3. Afficher le résultat. Ce qui conduit au programme suivant :

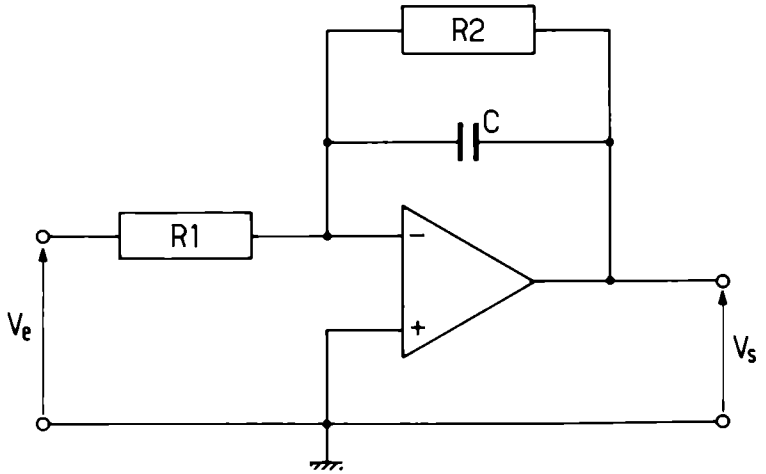
```
10 '
20 '          ************************
30 '          *                      *
40 '          * CONVERSION DEC-BIN   *
50 '          *                      *
60 '          ************************
100 INPUT"Donner la valeur du nombre ";
N
```

```
110    FORJ=7TO0STEP-1
120        B(J)=INT(N/2^J)
130        N=N-B(J)*2^J
140        PRINTB(J);
150    NEXTJ
160 PRINT
170 GOTO100
```

On constate que le programme ci-dessus a été écrit en décalant de quelques caractères l'ensemble des instructions de la boucle FOR... NEXT, afin de rendre le programme plus lisible. On peut parler alors *d'écriture STRUCTUREE.*

• *Tracé de la courbe de réponse en fréquence d'un filtre R-C.*

Le but de cette étude est d'analyser le comportement d'un filtre actif lorsque l'on fait varier la valeur de l'un des composants. Soit le filtre ci-dessous :



Le gain de ce filtre est :

$$A_V = \frac{V_S}{V_e} = -\frac{R_2/R_1}{\sqrt{1 + R^2_2\, C^2\, \omega^2}}$$

avec $\omega = 2.\pi.f$ f = fréquence en Hz.

On se propose d'étudier l'influence de la valeur de C dans la réponse en fréquence de ce filtre passe-bas du 1er ordre. On supposera $R_2 = 100 \cdot R_1 = 100\ k\Omega$.

La fréquence f prendra les valeurs multiples de 10, 20, 50 dans une gamme allant de 10 Hz à 500 kHz.

La courbe représentera le rapport $|A_V/A_{VO}|$ en décibels, soit :

$$20 * \log_{10} \left| \frac{A_V}{R_2/R_1} \right| = 20 * \log_{10} \left[\frac{1}{\sqrt{+ R^2{}_2 C^2 \omega^2}} \right]$$

$$= -10 * \log_{10}(1 + R^2{}_2 C^2 \omega^2)$$

en fonction de la fréquence f. (On portera en fait la valeur de $\log_{10} f$ en abscisse.)

On devra également faire apparaître sur l'écran la valeur de la fréquence de coupure $f_o = \frac{1}{2 \pi R_2 C}$.

Ce programme sera étudié en deux parties :
- 1re partie : calcul des coordonnées des points de la courbe.
- 2e partie : tracé de la courbe.

La deuxième partie sera étudiée dans les chapitres XIII et XIV consacrés aux dessins et graphiques.

On rappelle que $\text{Log}_{10} X = \underbrace{\text{Log}_e X}_{\text{LNX}} * \text{Log}_{10} e = 0{,}43429419 * \text{LNX}$

Donc $G_{dB} = -4{,}3429419 * LN(1 + (R_2 * C * 2 * \pi * f) \uparrow 2)$

On en déduit le programme suivant :

```
10 '          ************************
20 '          *                      *
30 '          * REPONSE D'UN FILTRE *
40 '          *                      *
50 '          ************************
60 '
70 '
80 '-------------CALCUL DU GAIN------
90 '
100 A$="###.##^^^^^":B$="##":C$="##^^^^^"
110 DIMG(15),F(15):E=4.3429419:P=3.14159
```

```
27:R2=1E5
120 INPUT"Donner la valeur de C : ";C
130 F0=1/(2*P*R2*C):CLS:PRINT"F0 = ";INT
(F0);"Hz"
140    FORI=1 TO 15
150        READF(I)
160        G(I)=-E*LOG(1+(R2*C*2*P*F(I))^2
)
170        PRINT"F(";:PRINTUSINGB$;I;:PRIN
T")=";:PRINTUSINGC$;F(I);:PRINT"     G(";
:PRINTUSINGB$;I;:PRINT")=";:PRINTUSINGA$
;G(I)
180    NEXTI
190 DATA10,20,50,100,200,500,1E3,2E3,5E3
,1E4,2E4,5E4,1E5,2E5,5E5
```

On remarquera que l'instruction REM peut être remplacée par une apostrophe.

Les résultats sont reproduits ci-dessous :

```
Donner la valeur de C: 1E-10
F0 =  15915 Hz
F( 1)= 1E+01     G( 1)=-12.56E-07
F( 2)= 2E+01     G( 2)=-64.59E-07
F( 3)= 5E+01     G( 3)=-42.70E-06
F( 4)= 1E+02     G( 4)=-17.08E-05
F( 5)= 2E+02     G( 5)=-68.51E-05
F( 6)= 5E+02     G( 6)=-42.83E-04
F( 7)= 1E+03     G( 7)=-17.11E-03
F( 8)= 2E+03     G( 8)=-68.04E-03
F( 9)= 5E+03     G( 9)=-40.88E-02
F(10)= 1E+04     G(10)=-14.45E-01
F(11)= 2E+04     G(11)=-41.15E-01
F(12)= 5E+04     G(12)=-10.36E+00
F(13)= 1E+05     G(13)=-16.07E+00
F(14)= 2E+05     G(14)=-22.01E+00
F(15)= 5E+05     G(15)=-29.95E+00
```

CHAPITRE IX **Les chaînes de caractères**

1. REPRESENTATION INTERNE

Lorsque dans un programme on définit une variable chaîne, automatiquement, en fin de programme, sont rangés en mémoire :

- le séparateur indiquant une déclaration de chaîne : $30,
- le nom de la variable.

Puis sur 3 octets on trouve :

- la longueur (nb total de caractères) de la variable,
- l'adresse où elle est définie, codée sur 2 octets.

La longueur de la variable, étant codée sur un octet, ne peut donc dépasser 255 caractères.

Exemple : imaginons le programme ci-dessous.

```
10 CLS:SCREEN4,6,6
20 OPEN"O",#1,"LPRT:(40)"
30 A$="%    %":B$="%    %"
40 FORN=&H666F TO &H667F
50 PRINT#1,HEX$(N)+" "+HEX$(PEEK(N))
60 NEXTN
```

Résultats :

```
666F 0        6678 42
6670 0        6679 5
6671 0        667A 66
6672 30       667B 2F
6673 41       667C 40
6674 6        667D 4E
6675 66       667E 8F
6676 23       667F 4C
6677 30
```

Ce programme se range en mémoire à partir de l'adresse hexadécimale &H65F5 jusqu'à &H66DC et l'on trouve alors l'occupation mémoire suivante :

Adresse	Contenu hexa	Commentaires
6672	30 →	Indicateur de chaîne $
6673	41	Nom de la variable : A
6674	06	Longueur : 6 caractères
6675	66 }	Adresse où A$ est définie
6676	23 }	&H6623
6677	30 →	
6678	42	Variable B
6679	05	Longueur 5 caractères
667A	66 }	Adresse &H662F
667B	2F }	

2. *OPERATEUR CONCATENATION*

Si l'on veut fabriquer une chaîne de plus de 255 caractères une méthode existe qui consiste à découper cette chaîne en autant de sous-chaînes que nécessaire puis à utiliser la somme (ou concaténation) de celles-ci.

Exemple : si longueur de A$ = 200
si longueur de B$ = 100
on peut toutefois écrire : PRINT A$ + B$

Le signe + représente la concaténation (mise bout à bout) des deux chaînes A$ et B$.

3. *OPERATEURS RELATIONNELS*

Les opérationnels relationnels =, <, >, sont utilisables avec les chaînes de caractères.

Les deux premières lettres sont comparées, puis, si elles sont identiques, les deux suivantes...

Une chaîne est inférieure à une autre dès que l'une des deux lettres de la comparaison est plus petite (c'est-à-dire plus près du A) que l'autre, ou si, à égalité, l'une des chaînes est plus courte.

Exemple :

« AFRIQUE » > « AFRICAIN »

↓ ↓

car q > c

par contre : « AFRICAIN » < « AFRICAINE », car plus court.

Ces opérateurs permettront de réaliser facilement un rangement par ordre alphabétique.

Exemple : Rangement et mise par ordre alphabétique de 8 variables.

```
1  '        ********************************
2  '        *                              *
3  '        * PROGRAMME DE TRI ALPHABE.    *
4  '        *                              *
5  '        ********************************
6  '
7  '
8  '--------LECTURE DES DONNEES---------
9  '
10 FOR N=1TO8
20      READA$(N)
30      PRINTA$(N)+" ";
40      NEXTN
42 '
44 '
46 '------COMPARAISON DES CHAINES------
48 '
50 B$(1)=A$(1)
60 FORI=2TO8
70    FORJ=1TOI-1
80       IFA$(I)<=B$(J)THENGOSUB1000:GOTO
100
90       NEXTJ
95    B$(I)=A$(I)
100 NEXTI
110 '
120 '
130 '------AFFICHAGE DES RESULTATS-----
```

```
140 '
150 '
200 PRINT:PRINT:PRINT
300 FOR S=1 TO 8
310   PRINTB$(S)+" ";
320 NEXTS
400 END
990 '
992 '
994 '----SOUS-PROGRAMME DE PERMUTATION-
996 '
998 '
1000 FORN=I TO J+1 STEP-1
1010   B$(N)=B$(N-1)
1020   NEXTN
1030 B$(J)=A$(I)
1040 RETURN
1990 '
1992 '
1994 '----RANGEMENT DES CHAINES--------
1996 '
1998 '
2000 DATA CHARLES,STEPHANE,ANDRE,MICHEL,
ALPHONSE,ETIENNE,ANATOLE,PAUL
```

Résultat :

```
ALPHONSE ANATOLE ANDRE CHARLES ETIENNE
MICHEL PAUL STEPHANE
```

4. *LES INSTRUCTIONS DE CHAINES*

ASC (A$)

Cette instruction renvoie, en décimal, le code ASCII du premier caractère de la chaîne A$.

CHR$ (Arg)

Cette instruction permet d'obtenir le caractère dont le code ASCII est donné en décimal par l'argument. (Voir plus loin, police des CHR$.)

CLEAR Arg

Cette instruction est nécessaire pour réserver en mémoire plus de 300 caractères de chaîne.

En résumé, l'instruction CLEAR a trois arguments utilisables simultanément et séparés par une virgule.

Le premier réserve de la place pour les chaînes.
Le second pour un sous-programme en assembleur.
Le troisième pour les caractères GR$.

Exemple :

```
CLEAR 400 ,     &H8000 ,     5
    ↓              ↓          ↓
400 octets chaîne  Fin du Basic  5 . GR$
                   et des variables
```

FRE (A$)

Cette instruction renvoie le nombre d'octets restant disponibles pour les chaînes.

HEX$ (Arg)

Cette instruction convertit en hexadécimal chaîne la valeur décimale de l'argument.

INSTR (Arg, A$, B$)

Cette instruction permet de rechercher la position de la sous-chaîne B$ dans la chaîne A$, à partir du caractère de rang indiqué par l'argument s'il est présent.

LEFT$ (A$, Arg)

Cette instruction permet d'obtenir une chaîne réalisée en ne conservant que les caractères à gauche de A$ en nombre égal à l'argument.

LEN (A$)

Cette instruction renvoie la valeur décimale de la longueur de la chaîne A$.

MID$ (A$, Arg, L)

Cette **instruction** permet d'extraire la chaîne de longueur L réalisée en ne conservant que les caractères de A$ à partir du rang donné par l'argument.

MID$ (A$, X, Y) CHAINE

Cette **fonction** permet de remplacer dans A$, à partir du Xe caractère, un nombre de caractères Y qui sont les Y premiers de CHAINE.

Exemple :

```
10 A$="ATOMIQUE"
20 PRINTMID$(A$,2,3)
30 MID$(A$,1,4)="PATH"
40 A$="A"+A$
50 PRINTA$
```

donnera

```
TOM
APATHIQUE
```

OCT$ (Arg)

Cette instruction donne une chaîne représentant la valeur octale de l'argument décimal.

RIGHT$ (A$, Arg)

Cette instruction permet d'obtenir une chaîne en ne conservant que les caractères à droite de A$ en nombre égal à l'argument.

STR$ (Arg)

Cette instruction fournit une chaîne représentant le nombre donné par l'argument.

VAL (A$)

Cette instruction donne la valeur décimale du nombre chaîne A$.

5. POLICE DE CARACTERES – CHR$ (1)

De Ø à 31 (valeurs décimales) on a l'équivalence suivante :

Ø	NUL	16	
1		17	Curseur clignotant
2	STOP	18	Répétition
3		19	
4		20	Arrêt du curseur
5		21	
6		22	Mode accentué
7	BEEP	23	
8	←	24	Efface la fin de la ligne
9	→ déplacement	25	
10	↓ du curseur	26	
11	↑	27	ESC
12	RAZ	28	INS
13	ENTREE	29	EFFACE
14	Semi-graphique	30	HOME
15	Alphanumérique	31	US

De 32 à 127, le programme ci-dessous permet d'obtenir l'équivalence nombre décimal ↔ caractère.

```
1 '
2 '
3 '          ***************************
4 '          *                         *
5 '          * POLICE DE CARACTERES    *
6 '          *                         *
7 '          ***************************
8 '
9 '
10 FORN=32 TO 127 STEP4
20     PRINTN;" "+CHR$(N);TAB(10);
30     PRINTN+1;" "+CHR$(N+1);TAB(20);
40     PRINTN+2;" "+CHR$(N+2);TAB(30)
50     PRINTN+3;" "+CHR$(N+3)
60     NEXTN
70 END
```

(1) Voir annexe 7.

32		33	!	34	"	35	#
36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+
44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3
52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;
60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C
68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K
76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S
84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[
92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c
100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k
108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s
116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{
124	|	125	}	126	~	127	

Remarque : dans les liaisons avec différentes imprimantes, certains codes ci-dessus (de Ø à 31) peuvent avoir des effets spéciaux.

Par exemple, avec l'imprimante à impact PR90.080, l'impression de CHR$(14) provoque le passage en mode double largeur et CHR$(15) ramène au mode normal.

6. *CODES SPECIAUX*

Les caractères GR$ définis par l'utilisateur peuvent être appelés par l'instruction CHR$ en respectant la correspondance suivante :

GR$ (N) = CHR$ (128 + N)

Exemple :

```
GR$(Ø) = CHR$(128)
GR$(1) = CHR$(129)...
```

Les codes des accents sont les suivants :

Accent grave 65	Accent aigu 66
Accent circonflexe 67	Tréma 72
Cédille 75	

Exemple : Pour écrire « été » il faut taper :

```
PRINT CHR$(22) + CHR$(66) + CHR$(101) + « t » + CHR$(22)
      + CHR$(66) + « e »
```

7. CARACTERES SEMI-GRAPHIQUES TELETEL

Une police de caractères aux normes Télétel est accessible par CHR$(14) suivi du code du caractère (voir tableau ci-dessous).

Chaque caractère est réalisé à partir d'un carré de 8 × 8 PL, selon le découpage donné par la figure (fig. IX-1).

Chaque case de ce carré à une valeur qui est une puissance de 2, de 2^0 à 2^5, ce qui permet d'obtenir 64 caractères de 32 à 63 puis de 96 à 127.

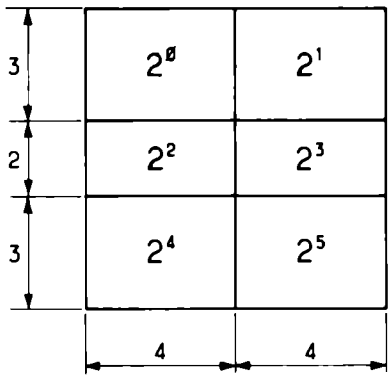


Fig. IX.1-b. – Détail d'un caractère Télétel.

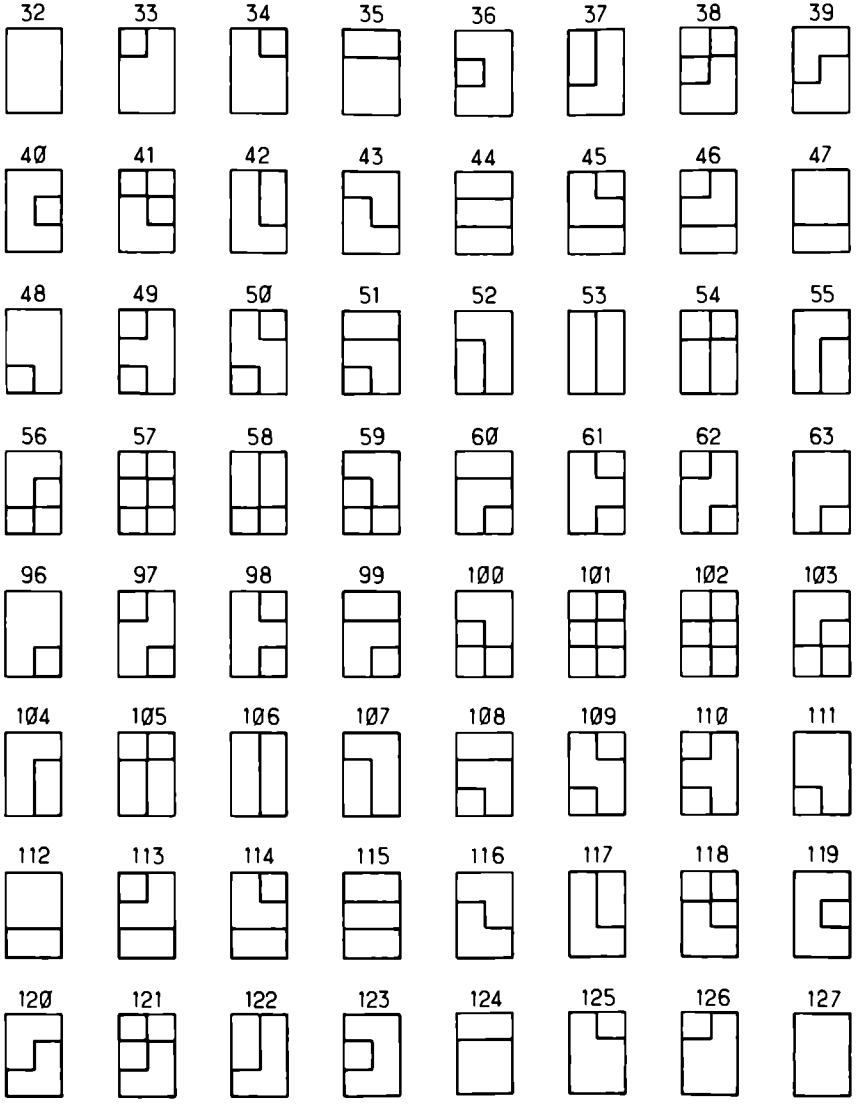


Fig. IX. 1-a. – Caractères semi-graphiques Télétel.

CHAPITRE X **Les tableaux de variables**

1. EXEMPLE INTRODUCTIF

Supposons que nous ayons rangé sur la cassette toutes les données d'un répertoire alphabétique, soit 5Ø noms de personnes, 5Ø prénoms, 5Ø adresses et 5Ø numéros de téléphone.

L'ensemble du fichier représente 2ØØ variables et, si nous voulons pouvoir travailler avec toutes ces variables simultanément, il faut pouvoir les ranger en mémoire sous des noms différents.

D'autre part, il est souhaitable de pouvoir traiter ces variables dans des boucles de 1 à 5Ø.

La solution idéale consiste donc à *indicer* chaque variable, c'est-à-dire à utiliser le même nom de variable suivi d'un numéro entre parenthèses.

Exemple :

NOM$(1) = « DURAND »	PRE$(1) = « JEAN »...
NOM$(2) = « DUBOIS »	PRE$(2) = « GEORGES »...
NOM$(3) = « LEPIED »	PRE$(3) = « CLAUDE »...
"	"
"	"
"	"
NOM$(5Ø) = « MAHUT »	PRE$(5Ø) = « MICHEL »...

2. DIMENSIONNEMENT – OCCUPATION

Le micro-ordinateur TO 7 doit réserver en mémoire la place nécessaire au rangement des variables.

S'il rencontre, dans un programme, une variable indicée, il réserve automatiquement la place nécessaire au stockage de 11 variables du même type, numérotées (indicées) de Ø à 1Ø.

Exemple :

```
3Ø NOM$(1)="DURAND"
```

⇒ l'ordinateur réserve la place pour
NOM$(Ø), NOM$(1), NOM$(2)... NOM$(1Ø)

Si l'indice dépasse 1Ø, il faut dimensionner la variable.

DIM

Cette instruction réserve en mémoire la place nécessaire au rangement des variables déclarées.

Exemple :

```
1Ø DIMNOM$(5Ø),PRE$(5Ø),ADR$(5Ø),TEL$(5Ø
)
```

Les instructions ci-dessus réservent la place pour 2Ø4 variables, NOM$(Ø)... NOM$(5Ø), PRE$(Ø)... PRE$(5Ø)...

Dans l'exemple précédent, on aurait pu identifier chaque personne par une fiche. Chaque fiche est numérotée de 1 à 5Ø.

Les variables sur les fiches sont repérées par des numéros.

S'il s'agit d'un nom, le numéro est 1
S'il s'agit d'un prénom, le numéro est 2
S'il s'agit d'une adresse, le numéro est 3
S'il s'agit d'un numéro de téléphone, le numéro est 4

Donc le prénom de la fiche 22 sera indentifié par :

FIC$ (22,2) = « PAUL »

Le nom de la fiche 21 par : FIC$(21,1) = « LABRANCHE »

On définit ainsi *un tableau* de variables à deux dimensions.

FICHE (X,Y)	Y = 1	Y = 2	Y = 3	Y = 4
X = 1	DURAND	JEAN	20 rue des Prés	0340619
X = 2	DUBOIS	GEORGES		
X = 3	LEPIED			
X = 4				

Fig. X-1. – Tableau à deux dimensions.

Ce tableau doit être dimensionné par l'instruction :

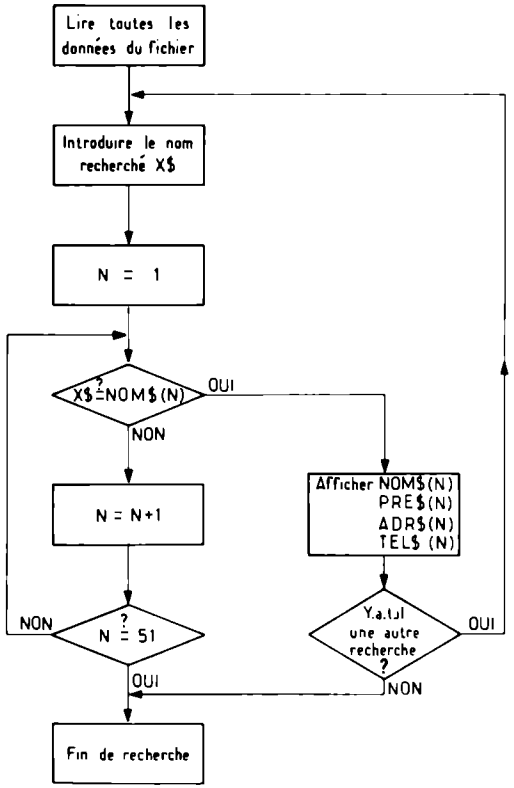
DIM FIC$ (5Ø,4)

On peut ainsi théoriquement définir des tableaux ayant jusqu'à 255 indices, chaque indice pouvant aller jusqu'à 32767, si la place en mémoire le permet.

Les valeurs de ces variables sont ou bien définies dans le programme, ou bien rangées en fin de mémoire, juste avant les GR$.

3. *APPLICATION*

Supposons que le fichier existe déjà sur cassette et examinons le problème qui consiste, à partir d'un nom donné par l'opérateur, à rechercher les coordonnées correspondantes.



Cette méthode évite, dans le cas d'une recherche répétitive, de rembobiner la cassette pour relire à chaque fois le fichier, ce qui prend beaucoup trop de temps. Ici le fichier n'est lu qu'une seule fois en début de programme et toutes les variables indicées sont rangées en mémoire.

Cette méthode n'est plus utile avec un lecteur de disquette.

```
1 '        ****************************
2 '        *                          *
3 '        * RECHERCHE D'UNE ADRESSE *
4 '        *                          *
5 '        ****************************
6 '
7 '
8 '
9 DIMNOM$(50),PRE$(50),ADR$(50),TEL$(50)
10 OPEN"I",#2,"REPER"
20 FOR N=1 TO 50
30      INPUT#2,NOM$(N),PRE$(N),ADR$(N),T
EL$(N)
40      NEXTN
50 INPUT"Nom recherche :",N$
60 N=1
70 IFN$=NOM$(N)GOTO110
80 N=N+1:IFN=51GOTO160
90 GOTO70
100 END
110 PRINTNOM$(N)+" "+PRE$(N)
120 PRINTADR$(N)
130 PRINT"                    TEL : "+TEL$(N)
140 PRINT:PRINT:PRINT"Cherchez-vous quel
qu'un d'autre ?"
150 A$="":A$=INKEY$:IFA$=""GOTO150ELSEIF
A$="O"GOTO50ELSECLOSE#2:END
160 PRINT"Erreur. Ce nom n'est pas dans
le repertoire.":CLOSE#2:END
```

La création du fichier est étudiée plus loin.

CHAPITRE XI **Les instructions de périphériques**

1. LE LIGHT-PEN

• *Principe de fonctionnement*

Le crayon optique contient deux appareils totalement indépendants :

– un interrupteur, situé dans le nez du crayon ;
– un phototransistor, situé à l'intérieur du crayon.

Ce dernier, lorsqu'il reçoit suffisamment de lumière, laisse passer le courant électrique venant du TO 7 ou, au contraire, bloque ce courant s'il est en face d'une zone sombre.

Chaque point de l'écran étant allumé (ou éteint) à un instant donné, tout l'écran est balayé point par point à une vitesse telle que, persistance rétinienne aidant, on a l'impession de les voir tous éclairés simultanément.

Placé face à un point (ou un paquet de points) de l'écran, le crayon optique indiquera donc si ce point est allumé ou éteint. Puisque c'est le TO 7 lui-même qui gère l'écran, il connaît à chaque instant l'état et la position des points sur l'écran, donc la position du crayon optique.

Nous allons donc pouvoir disposer de fonctions Basic combinant les propriétés du light-pen et les possibilités graphiques de gestion de l'écran.

• *Les instructions Basic*

INPEN C,L

Cette instruction renvoie immédiatement les coordonnées du point graphique de l'écran placé face au crayon optique.

C est le numéro de colonne, compris entre Ø et 319.

L est le numéro de ligne compris entre Ø et 199. Le numéro de colonne C correspond toujours au numéro de la colonne du milieu d'un caractère, donc varie par pas de 8 à partir de la première valeur 3.

Si le crayon optique ne détecte pas de lumière, il renvoie X = −1 et Y = −1.

INPUTPEN C,L

Cette instruction fonctionne comme la précédente, seulement dans le cas où l'interrupteur du crayon optique est enfoncé.

PEN Z0, Z1,...

Cette instruction permet de définir 8 zones rectangulaires sur l'écran, de la zone 0 à la zone 7.

Une zone est définie de la façon suivante :

Z0 ≡ N° de zone ; coordonnées de la zone.

Si la zone est de la dimension d'un caractère, les coordonnées sont du type caractère et sont donc constituées du numéro de colonne C compris entre 0 et 39 et du numéro de ligne L compris entre 0 et 24.

Exemple : PEN0 ; (1,3), voir figure XI-1.

Si la zone est supérieure à un caractère, on définit ses coordonnées en mode graphique en indiquant les coordonnées colonne et ligne des deux points extrêmes de sa diagonale.

Exemple : PEN1 ; (5, 12) – (10,24), voir figure XI-2.

Si on veut supprimer une zone, il suffit d'indiquer son numéro par l'instruction PEN.

Exemple :

PEN 1 supprime la zone 1
PEN supprime toutes les zones.

Voir figures XI-1 et XI-2.

ONPENGOTO NL

Cette instruction agit comme ON GOTO mais cette fois la variable est le numéro de la zone qui sera détectée par le crayon optique.

La séquence d'instruction sera donc la suivante :

1. Définir les zones : PEN0 ; (1,3), 1 ; (7,9)
2. Lire la position du crayon : INPUT PEN C,L
3. Se brancher vers le n° de ligne en fonction de la zone pointée : ON PEN GOTO 100, 200

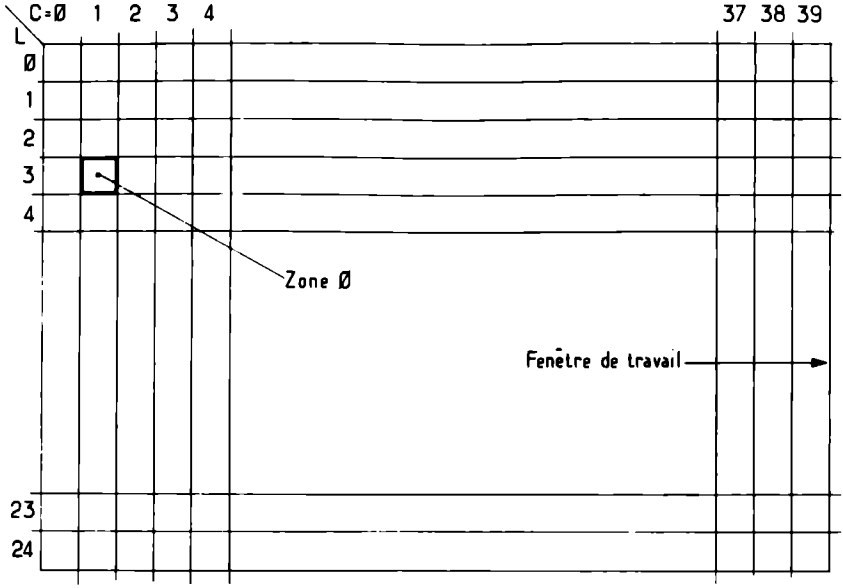


Fig. X-1. – Tableau à deux dimensions.

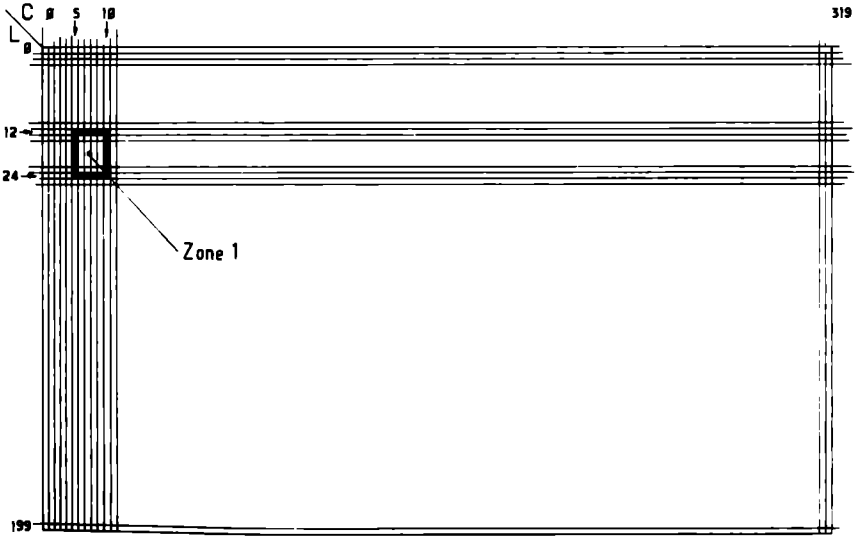


Fig. XI-2.

ONPENGOSUB NL1, NL2

Cette instruction agit comme la précédente, mais en appelant cette fois un sous-programme dépendant de la zone pointée.

PTRIG

Cette instruction est une variable qui peut prendre la valeur –1 si l'interrupteur du crayon optique est actionné, ou la valeur Ø dans le cas contraire.

On l'utilisera le plus souvent dans des instructions de test du type : IF PTRIG = Ø THEN...
ou bien IF PTRIG = –1 THEN...

• ***Exemple d'application***

Voici un petit programme qui vous permet de faire un dessin sur l'écran tant que vous appuyez le crayon sur celui-ci.

```
PROGRAMME DE DESSIN

10 SCREEN4,6,6:CLS
20 IF PTRIG = 0 GOTO20
30 INPEN X,Y:IF X<0 OR Y<0 GOTO 30 ELSE
PSET(X,Y)
40 IF PTRIG =0 GOTO20
50 INPEN X,Y:IF X<0 OR Y<0 GOTO 50 ELSE
LINE-(X,Y)
60 GOTO40
70 END
```

2. *LES MANETTES DE JEUX*

Les manettes de jeux comprennent un bouton-poussoir et un levier. Le TO 7 peut détecter :

– l'action sur le bouton,
– la position du levier par 1/8e de tour.

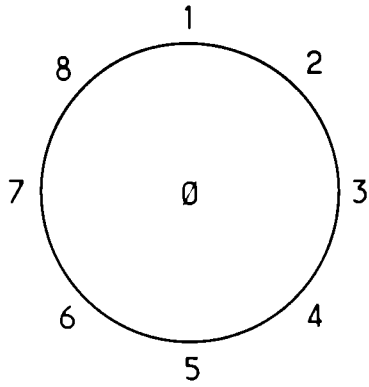


Fig. XI.3.

STICK (N)

Cette instruction renvoie la position du levier de la manette numéro N.

Si le levier est vertical, STICK vaut Ø.
Si le levier est à midi, STICK vaut 1, etc.

Voir la figure XI-3 pour le codage complet.

STRIG (N)

Cette instruction, analogue à PTRIG, renvoie Ø si le bouton de la manette N n'est pas actionné, ou –1 si le bouton est actionné.

3. LA MUSIQUE

BEEP

Cette instruction déclenche une petite sonnerie, celle que l'on entend quand on tape sur le clavier, et que l'on peut obtenir également par CHR$(7).

PLAY A$

Cette instruction permet de jouer l'air de musique défini par la chaîne A$ de la façon suivante :

La chaîne contient la suite des notes que l'on veut jouer.

Chaque note est définie par deux lettres : DO – RE – MI – FA – SO – LA – SI et, éventuellement, les symboles dièse # ou bémol b.

Chaque note peut être choisie dans une octave parmi cinq. L'octave 1 (01) étant la plus grave et 05 la plus aiguë. Une octave couvre les notes de DO, RE... à SI, SI #.

La longueur de chaque note doit être précisée. Cette longueur ou durée peut varier de 1 à 96 par bond de 1. La noire a une longueur L24, donc la croche L12, la blanche L48...

On peut également définir l'amortissement de chaque note, entre Ø et 255.

Si l'amortissement est nul (AØ), le son est continu.

Le rythme de l'ensemble, ou Tempo, peut varier de 1 à 255, 1 étant le rythme maximum correspondant donc à l'exécution la plus rapide.

On dispose en outre des silences P considérés comme des notes et suivant donc les règles ci-dessus.

Exemple : comment jouer « au clair de la lune » :

```
PLAY «DODODOREL48MIREL24DOMIREREL48DO»
```

Au départ, par défaut, on considère :

- Attaque AØ
- Longueur L24
- Tempo T5
- Octave 04

On peut d'autre part manipuler la chaîne à jouer comme n'importe quelle variable chaîne.

Exemple : Programme permettant d'écrire la note en la jouant.

```
1 '          ************************
2 '          *                      *
3 '          * CREATION MUSICALE    *
4 '          *                      *
5 '          ************************
6 '
7 '
8 '--Initialisation, dessin d'une note--
9 '
10 CLEAR,,1:CLS:DIMNO$(20)
20 DEFGR$(0)=24,126,126,255,255,126,126,
```

```
20
22 '
24 '
26 '------Dessin de la Portee----------
28 '
30 FORL=4TO68STEP16
40    LINE(0,L)-(319,L)
50    NEXTL
60 X=0:Y=10:N=1
70 IFN=21THENGOTO120
72 '
74 '
76 '-----Lecture de la note-----------
78 '
80 NO$(N)=INPUT$(2)
90 RESTORE200
92 '
94 '
96 '-Recherche de la note dans la table-
98 '
100 READN$
110 IFNO$(N)=N$THENLOCATEX,Y:PRINTGR$(0)
:PLAY"L96"+N$:X=X+2:Y=10:N=N+1:GOTO70 EL
SE Y=Y-1:GOTO100
112 '
114 '
116 '---Lecture de la Partition--------
118 '
120 FORN=1TO20
130    PLAY"L96"+NO$(N)
140    NEXTN
150 END
192 '
194 '
196 '-------Table des notes-------------
200 DATADO,RE,MI,FA,SO,LA,SI,PP
```

4. LES INSTRUCTIONS DE FICHIER

• *Description de fichier*

Un fichier est un ensemble d'informations que l'on écrit ou qu'on lit sur un périphérique. Ce fichier possède en général un nom.

Suivant la nature des informations, on distingue plusieurs types de fichier :

- les fichiers de programmes,
- les fichiers de données,
- les fichiers binaires.

Pour accéder à un fichier, il faut toujours l'ouvrir. Puis il faut le fermer quand on a terminé de l'utiliser.

• *Les types de périphériques*

L'ordinateur TO 7 possède les périphériques suivants :

- le lecteur-enregistreur de cassettes CASS
- l'écran SCRN
- l'imprimante parallèle LPRT
- le clavier KYBD
- l'interface série COMM

Lorsque l'on travaille avec l'imprimante parallèle, ou avec l'interface série, on peut définir un certain nombre d'options.

- Pour l'imprimante, on peut indiquer le nombre de caractères par ligne.
- Pour l'interface série, on précise la vitesse de transmission.

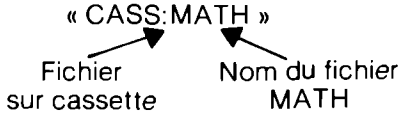
1 pour 11Ø bits/seconde, soit	11Ø bauds
2	3ØØ bauds
3	6ØØ bauds
4	1 2ØØ bauds
5	2 4ØØ bauds
6	4 8ØØ bauds

Puis on précise si l'émission se fait sur 7 ou 8 bits, et enfin on indique le nombre de caractères/ligne.

Les options seront donc représentées par un chiffre placé entre parenthèses à la suite du nom du périphérique, et en le séparant de ce dernier par : (deux points).

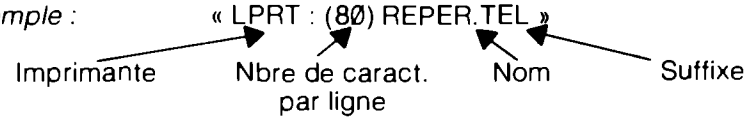
Suivra l'ensemble, le nom du fichier éventuellement. Le tout placé entre guillemets constitue le descripteur de fichier.

Exemple :



Le nom du fichier ne doit pas dépasser 8 caractères sans point ni virgule et peut être suivi d'un suffixe de 3 caractères, séparés des 8 premiers par un point.

Exemple :



Lorsqu'on ne précise pas le périphérique choisi, c'est l'écran qui est retenu pour l'instruction LIST, et le lecteur-enregistreur pour les autres instructions.

- ***Les instructions***

CLOSE # N°

Cette instruction ferme le fichier de numéro N. L'instruction CLOSE, utilisée seule, ferme tous les fichiers préalablement ouverts.

EOF (N°)

Cette instruction indique la fin d'un fichier. Elle prend la valeur Ø si la fin de fichier n'est pas atteinte, et la valeur −1 dans le cas contraire. Elle est en général utilisée dans des tests.

INPUT # N°, Var1, Var2

Cette instruction lit séquentiellement, sur le fichier de N° indiqué, les variables qui suivent comme pour une instruction INPUT.

Un exemple d'utilisation de cette instruction a été donné dans le cas du « Répertoire téléphonique ».

INPUT$ (Arg), # N°

Cette instruction permet de lire une entrée, sur le fichier de N° donné, comportant un nombre de caractères défini par l'argument.

LINE INPUT # N°, Var$

Cette instruction permet de lire, sur le fichier de N° indiqué, la variable chaîne.

LIST

Cette instruction permet de lister sur l'écran la totalité du programme en mémoire si celui-ci n'est pas verrouillé.

On peut lister la partie du programme comprise entre les lignes NL1 et NL2 en ordonnant :

LIST NL1-NL2

Si on supprime NL1 on liste tout le programme jusqu'à la ligne NL2.
Si on supprime NL2 on liste tout le programme depuis NL1.

L'instruction LIST NL permet de ne lister que la ligne de numéro NL.

LIST « CASS : NOM », NL1-NL2

Cette instruction permet la sauvegarde sous forme d'un listing d'un programme appelé NOM, des lignes NL1 à NL2.

LIST «LPRT : (NC) », NL1-NL2

Cette instruction permet la sortie vers le périphérique imprimante parallèle des lignes NL1 à NL2 d'un programme, à raison de NC caractères par ligne. Par défaut, NC = 4Ø.

LIST « COMM : (V), NOM », NL1-NL2

Cette instruction permet le transfert, sur la ligne de communication série RS232, des lignes NL1 à NL2 du programme NOM.

Le code de V sélectionne vitesse, nombre de bits (7 ou 8) par caractère et nombre de caractères/ligne.

LOAD « NOM », R

Cette instruction permet de charger en mémoire un programme sauvegardé sur la cassette avec le même NOM.

Si R est présent, il y a lancement automatique en mode RUN.

LOAD» seul charge le premier programme rencontré.

MOTOR ON (OFF)

Ces instructions provoquent le défilement (ou l'arrêt) de la bande du lecteur enregistreur de cassettes si celui-ci était en mode PLAY ou RECORD.

MERGE « NOM »

Cette instruction permet de charger en mémoire un programme sauvegardé sur cassette sous le même NOM sous forme ASCII, en le mélangeant au programme déjà en mémoire.

Si deux lignes ont le même numéro, c'est celle qui vient du lecteur qui remplace celle déjà en mémoire.

Exemple :

Programme en mémoire :	Programme fichier « MAO »
1Ø CLS : SCREEN Ø,7,7	2Ø PRINT « NUMERO 2 »
2Ø PRINT « NUMERO 1 »	3Ø END

On fait MERGE « MAO » et on obtient en mémoire :

```
1Ø CLS : SCREEN 0,7,7
2Ø PRINT « NUMERO 2 »
3Ø END
```

L'option R (Run) est autorisée comme pour LOAD.

OPEN « O » / « I », # N°, « PERIPH : (V), NOM »

Cette instruction ouvre le fichier numéro N°, sur le périphérique PERIPH, avec l'option V. Le fichier est nommé NOM.

Si OPEN « O » on établit la liaison TO 7 → PERIPH
Si OPEN « I » PERIPH → TO 7.

Le numéro du fichier doit être compris entre 1 et 16.

PRINT # N°, Var, Var

Cette instruction écrit sur le fichier N° les valeurs des variables indiquées en les séparant par un enregistrement du « Retour chariot .

PRINT # N° USING...

Cette instruction fonctionne comme PRINTUSING mais cette fois l'écriture se fait sur le fichier de numéro N°...

POS (N°)

Cette instruction donne la position du curseur dans la ligne en cours du fichier N°...

SAVE « NOM »

Permet la sauvegarde du programme NOM sur le lecteur enregistreur.

Si on ajoute l'option **,A**, cette sauvegarde se fait sous forme ASCII de façon à pouvoir être utilisée par MERGE.

Si on ajoute l'option **,P**, cette sauvegarde se fait sous forme protégée, ce qui empêche le listage et la recopie à partir de la mémoire.

SKIPF « NOM »

Cette instruction permet de sauter le fichier NOM sans l'enregistrer et de se placer immédiatement après.

Elle est très utile pour positionner la bande avant un enregistrement.

WAIT N°, I, J

Cette instruction arrête l'exécution du programme et teste l'état du périphérique ouvert sous le numéro de fichier N°.

Le programme n'est repris en séquence que lorsque la fonction logique ((Etat du PERIPH.) · I) + J est vraie.

- ***Cas d'un fichier binaire***

Les informations binaires (programme en assembleur, mémoire d'écran, GR$...) peuvent être stockées sur cassette et relues grâce aux instructions suivantes :

LOADM « NOM », OFS, R

Cette instruction permet le chargement en mémoire du fichier binaire NOM. Si OFS est absent, le chargement est implanté aux adresses qui étaient siennes lors de la sauvergarde. Si OFS est présent, l'implantation est décalée d'un offset égal à la valeur de OFS.

R est l'option RUN.

SAVEM « NOM », Adr1, Adr2, Adr3

Cette instruction permet la sauvegarde sur cassette du contenu mémoire de Adr1 jusqu'à Adr2, sous forme d'un fichier binaire appelé « NOM ».

Adr3 est l'adresse d'exécution, dans le cas d'un programme en assembleur.

Adr3 = Ad1 dans le cas de sauvegarde des GR$.

Adr3 = &HØØØØ dans le cas de sauvegarde du contenu de la mémoire d'écran.

5. *PROGRAMMES D'APPLICATION*

• *Création d'un fichier de données : le répertoire*

Dans le chapitre précédent, nous avions utilisé un fichier existant comprenant les noms, prénoms, adresses et téléphone de 5Ø personnes, pour faire une recherche d'adresse.

Ce fichier était ouvert en lecture avec le N° 2 :

1Ø OPEN « I », # 2, « REPER »

Puis les informations étaient rentrées quatre par quatre :

3Ø INPUT # 2, NOM$ (N), PRE$ (N), ADR$ (N), TEL$ (N)

Enfin le fichier 2 était fermé lignes 15Ø ou 16Ø par CLOSE # 2.

Pour créer ce fichier, on peut utiliser le programme suivant :

```
1  '            ****************************
2  '            *                          *
3  '            * CREATION DU REPERTOIRE  *
4  '            *                          *
5  '            ****************************
6  '
7  '
8  '------------Ouverture du fichier------
9  '
10 OPEN"O",#1,"CAS:REPER"
12 '
14 '
```

```
15 '---------Lecture des donees---------
18 '
20 FORN=1TO50
30     INPUT"NOM :",N$
40     INPUT"PRENOM :",P$
50     INPUT"ADRESSE :",A$
60     INPUT"TELEPHONE :",T$
62 '
64 '
66 '---------Ecriture sur le fichier----
68 '
70     PRINT#1,N$,P$,A$,T$
80     NEXTN
82 '
84 '
86 '---------Fermeture du fichier-------
88 '
90 CLOSE#1
100 END
```

- ***Dessin sur l'écran et sauvegarde du dessin***

On définit tout d'abord une zone de la dimension d'un caractère dans la case (X = 39, Y = 24) en bas et à droite de l'écran. Si le crayon optique pointe cette case, alors on sort du programme de dessin proprement dit pour sauver ce dessin sur cassette sous forme d'un fichier binaire.

On sauvegarde d'abord la forme du dessin, puis la couleur des points, ces informations étant stockées indépendamment en mémoire.

```
1 '             **************************
2               *                        *
3               * DESSIN SUR L'ECRAN     *
4 '             *                        *
5               **************************
6
7
8 '-----Effacement de l'ecran et Placeme
nt de la case 39,24----------------------
```

```
9 '
10 CLS
15 PSET(39,24)"."
16 '
17 '
18 '---------Definition de la zone 0-----
19 '
20 PEN0:(39,24):ONPENGOTO100
22 '
24 '
26 '--Lecture et Placement d'un Point--
28 '
30 INPUTPENC1,L1
40 PSET(C1,L1)
50 IFPTRIG=0GOTO20
52 '
54 '
56 '-----Lecture d'un Point et trait-----
58 '
60 INPUTPENC2,L2
70 IFC2<0ORL2<0GOTO30
80 LINE-(C2,L2)
90 IFPTRIG=0GOTO30ELSEGOTO60
92 '
94 '
96 '-----Sauvegarde du dessin-----------
98 '
100 POKE&HE7C3,1
101 '
102 '
103 '-----En &HE7C3 se trouve le Port C
de l'ACIA 6846 qui Permet entre'autre de
selectionner la memoire de forme d'un
dessin (Par mise a 1), ou la memoire de
104 'couleur de ce dessin (Par mise a 0)
.-----------------------------------------
105 '
106 '
107 '
```

```
110 SAVEM"DESSIN",&H4000,&H5F40,&H000
120 POKE&HE7C3,0
130 SAVEM"COULEUR",&H4000,&H5F40,&H000
```

La recopie du dessin de la cassette vers l'écran sera également faite en deux temps : tout d'abord la forme, puis la couleur.

```
1  '        ************************
2  '        *                      *
3  '        * RECOPIE D'UN DESSIN *
4  '        *                      *
5  '        ************************
6  '
7
8  '----Recuperation du dessin----------
9  '
10 POKE&HE7C3,1
20 LOADM"DESSIN"
22 '
24 '
26 '----Recuperation des couleurs-------
28 '
30 POKE&HE7C3,0
40 LOADM"COULEUR"
```

- ***Création musicale***

Ce programme réserve tout d'abord la place pour un GR$ et définit ce GR$, qui sera une note noire.

On peut rentrer par le clavier jusqu'à 20 notes.

Après avoir tracé une portée, on tape la note que l'on veut jouer : DO, RE, MI, FA, SO, LA, SI, PP.

On remarque le SOL sera tapé SO et le silence PP.

Chaque note tapée est jouée et placée sur la portée. Lorsque 20 notes ont été tapées, toute la partition est rejouée.

CHAPITRE XII **Les autres fonctions**

1. ECRITURE ET LECTURE EN MEMOIRE

Lorsqu'on le souhaite, on peut écrire des informations binaires, sous forme d'un octet, soit en mémoire vive (RAM) soit dans un registre de circuit d'interface pour périphérique.

Dans un cas comme dans l'autre, il faut connaître l'adresse de la mémoire ou du registre (Adr) pour pouvoir y écrire l'octet (Oct).

POKE Adr, Oct

Cette instruction permet de ranger l'octet Oct à l'adresse Adr.

On peut également lire le contenu d'une adresse de mémoire (morte ou vive) ou d'un registre.

PEEK (Adr)

Cette instruction prend la valeur (décimale) du contenu de l'adresse. Cette valeur peut être testée ou affectée à une variable.

Exemples d'utilisation

a) Comme pratiquement tous les microprocesseurs, le 6809 lors de son initialisation (Reset) va chercher l'adresse de départ du programme d'initialisation en &HFFFE, &HFFFF. Donc, pour le réinitialiser par programme, il faut faire :

```
AD = PEEK (&HFFFE) * 256 + PEEK (&HFFFF)
EXEC AD
```

b) Le contrôle du clavier est fait par un circuit de la famille 6800, le 6846 qui, outre 2048 octets de mémoire morte, contient un temporisateur 16 bits programmable et un port d'entrée/sortie 8 bits, avec ligne de contrôle. C'est ce port, qui se trouve à l'adresse &HE7C3, qui contrôle la mémoire d'écran (forme ou couleur) ainsi que le mode majuscules/minuscules par son bit b_3.

– Pour passer en mode majuscules, faire :

```
M = PEEK (&HE7C3) AND247
POKE &HE7C3, M
```

mise à 0 de b_3

– Pour passer en mode minuscules, faire :

```
M = PEEK (&HE7C3) OR8          mise à 1 de b3
POKE &HE7C3, M
```

c) *Implantation en mémoire d'un programme de temporisation écrit en langage source 6809*

La page Ø du moniteur occupe de la mémoire de l'adresse &H6ØØØ à &H6ØE3. Il reste donc de la place en RAM de &H6ØE4 à &H6ØFF pour loger un programme en assembleur. Sachant que l'octet de poids fort du temporisateur se trouve en &HE7C6, on peut établir un programme précis au $1/10^e$ de seconde en plaçant en &H6ØE4 la valeur de la temporisation (en 1/10 s) et en démarrant le programme d'assemblage en &H6ØE5.

Ce programme réalise deux tests successifs sur le contenu de &HE7C6 (temporisateur) :

– 1er test : passage à 1

– 2e test : passage à Ø.

La durée des deux tests, soit un cycle, représente environ 1/10 de seconde. On décrémente ensuite la valeur de la temporisation placée en &H6ØE4 à l'aide d'une décrémentation indexée utilisant le compteur programme comme index. Puis on procède à un test pour voir si le contenu de &H6ØE4 est nul :

– Si oui, on quitte ce sous-programme,
– Si non, on se rebranche en &H6ØE5.

Programme source ↓

```
&H60E4
&H60E5   7D E7 C6     TST E7 C6
&H60E8      27 FB     BEQ FB      branchement en 60E5
&H60EA   7D E7 C6     TST E7 C6
&H60ED      26 FB     BNE FB      branchement en 60EA
&H60EF   6A 8C F2     DEC F2, CP  décrémente 60E4
&H60F2      26 F1     BNE F1      branchement en 60E5
&H60F4         39     RTS         retour de S.P.
```

Une fois implanté en mémoire à l'aide du programme ci-dessous, ce programme source pourra être sauvé sur cassette comme fichier binaire.

```
1 '     ************************************
2 '     *                                  *
3 '     * TEMPORISATION PAR ASSEMBLEUR *
4 '     *                                  *
5 '     ************************************
6 '
7 '
8 '-----------Ouverture de la boucle---
9 '
10 FORN=1TO16
12 '
14 '
16 '-----------Lecture d'un octet------
20    READX$
22 '
24 '
26 '-----Conversion Hexadeimal-Decimal-
28 '
30    H$=LEFT$(X$,1)
40    L$=RIGHT$(X$,1)
50    H=ASC(H$)
60    IFH>64THENH=H-55ELSEH=H-48
70    L=ASC(L$)
80    IFL>64THENL=L-55ELSEL=L-48
90    X=H*16+L
100   POKE&H60E4+N,X
110   NEXTN
112 '
114 '
116 '-----Ecriture du Programme---------
118 '
120 DATA  7D,E7,C6,27,FB,7D,E7,C6,26,FB,
6A,8C,F2,26,F1,39
192 '
```

```
194 '
196 '---Verification de l'implantation--
198 '
200 FORN=&H60E5 TO &H60F4
210   PRINTHEX$(PEEK(N))+" ";
220   NEXTN
300 END
```

Les lignes 200, 210 et 220 permettent la vérification de l'implantation.

Pour exécuter une temporisation procéder comme suit :

Lancement d'une temporisation de 5 s (50. 1/10 s)

POKE &H60E4, 50
EXEC &H60E5 lance une temporisation de 5 s.

VARPTR (Var)

Cette instruction renvoie l'adresse du premier octet de la variable indiquée. Donc :

– pour les variables numériques entières, cette instruction renvoie l'adresse de l'octet de poids fort. L'octet de poids faible se trouve en VARPTR (Var) + 1,

– pour les variables numériques de simple et double précision, cette instruction renvoie l'adresse de l'exposant, les octets suivants formant la mantisse,

– pour les variables chaîne, cette instruction renvoie l'adresse de l'octet contenant la longueur de la chaîne, les deux octets suivants contenant le poids fort et le poids faible de l'adresse du premier caractère de la variable.

Exemple :

10 A$ = « BONJOUR » : N = 1

L'adresse du 1er caractère « B » de A$ est :

AD = PEEK (VARPTR (A$) + 1) * 256 + PEEK (VARPTR (A$) + 2)

Donc l'instruction
PRINT PEEK (AD) renvoie B
PRINT PEEK (AD + 1) renvoie O, etc.

2. GESTION DES ERREURS

Lorsqu'une erreur est détectée dans un programme, celui-ci cesse le travail en cours et renvoie sur l'écran un message d'erreur, indiquant le type de l'erreur par un numéro de code, et le numéro de la ligne où s'est produite l'erreur.

Les instructions suivantes permettent de créer des sous-programmes de gestion des erreurs, évitant ainsi la procédure ci-dessus.

ERL

C'est une variable égale au N° de la ligne où s'est produite une erreur.

ERR

C'est une variable égale au numéro de code de l'erreur.

ERROR code

Cette instruction placée dans un programme agit exactement comme si l'erreur de numéro indiqué par code était détectée.

En temps normal, elle provoque donc l'apparition d'un message d'erreur sur l'écran.

Dans le cas où un sous-programme de gestion aura été implanté, cette instruction entraîne l'appel de ce sous-programme.

ON ERROR GOTO N°L

Cette instruction, placée en début de programme, indique qu'un sous-programme de gestion des erreurs a été implanté en N° L et que l'on doit s'y rendre dès qu'une erreur est détectée.

RESUME

Cette instruction doit obligatoirement être placée en fin du sous-programme de gestion des erreurs. Elle permet le retour à la ligne où s'est produite l'erreur.

RESUME NEXT

Permet le retour à la ligne qui suit celle où s'est produite l'erreur.

RESUME N° L

Permet le retour au numéro de ligne indiqué.

Exemple d'utilisation

Lors d'une utilisation de l'imprimante comme fichier de sortie, si l'opérateur oublie de mettre l'imprimante sous tension, le TO 7 renvoie un message d'erreur. On y remédie comme suit :

```
1 '         ********************************
2 '         *                              *
3 '         * ECRITURE SUR L'IMPRIMANTE *
4 '         *                              *
5 '         ********************************
6 '
7 '
8 '-----Gestion de l'erreur-----------
9 '
10 ONERRORGOTO100
12 '
14 '
16 '-----Ouverture du fichier----------
18 '
20 OPEN"O",#1,"LPRT:(80)"
30 PRINT"Vous pouvez taper votre texte"
32 '
34 '
36 '--Lecture/Ecriture d'un caractere--
38 '
40 A$="":A$=INKEY$:IFA$=""THENGOTO40
50 PRINT#1,A$;:PRINTA$;
60 IFA$=CHR$(13)THENPRINT#1,CHR$(13):PRI
NTCHR$(13)
70 GOTO40
92 '
94 '
96 '-----Gestion de l'erreur-----------
98 '
```

```
100 IFERR=53THENPRINT"Mettez l'imprimant
e sous tension."
110 PRINT"Est-ce fait ?"
120 X$=INPUT$(1)
130 IFX$="O"THENRESUME
140 IFX$="N"THENEND
150 GOTO200
200 PLAY"DOMIDOPPPDOMIDO"
210 GOTO120
```

3. LES COMMANDES BASIC

AUTO NL, PAS

Cette commande permet une numérotation des numéros de lignes lors de la réalisation d'un programme. Celui-ci commencera à la ligne NL, les lignes étant numérotées de PAS en PAS.

Par omission NL et PAS valent 10.

On sort de la numérotation automatique par action simultanée sur les touches CNT et C.

CONT

Cette instruction permet de redémarrer un programme là où on l'avait stoppé par action sur CNT C ou lors de l'exécution d'un STOP.

DELETE NL1-NL2

Cette instruction efface du programme toutes les lignes comprises entre NL1 et NL2 incluses.

GOTO NL

Cette instruction permet le lancement d'un programme à la ligne NL sans réinitialiser les variables.

LIST

Cette instruction permet de lister un programme sur l'écran. On ne peut visualiser :

– qu'une ligne par LIST N° L,
– que jusqu'à une ligne par LIST – N° L,
– que depuis une ligne par LIST N° L –,
– qu'entre deux lignes par LIST N° L1 – N° L2.

Pour lister sur l'imprimante, il faut donner le nom de ce périphérique et, en option, le nombre de caractères par ligne.

Exemple :

LIST « LPRT : (80) », 10 – 100

NEW

Cette instruction efface le programme en mémoire. En fait, cette instruction remet à 0000 le pointeur de ligne qui se trouve en &H65F5 et &H65F6.

RUN

Cette instruction lance un programme à la première ligne en réinitialisant toutes les variables. On peut éventuellement donner le numéro de ligne de départ.

Exemple : RUN 100

RUN peut être également utilisé en remplacement de LOAD « NOM »,R. Dans ce cas, il charge le fichier présent sur le périphérique désigné, puis le lance comme un RUN de commande directe.

L'option, R (exemple : RUN « TOTO », R) permet le maintien des fichiers à l'état ouvert.

SAVE

Voir « Les instructions de périphériques ».

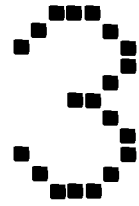
TRON

Cette instruction, utile pour la mise au point des programmes, permet l'affichage sur l'écran des numéros des lignes exécutées pendant qu'un programme tourne.

TROFF

Cette instruction supprime l'action de TRON.

Page blanche



Dessins et graphiques

CHAPITRE XIII **Les zones graphiques en RAM**

Comme nous l'avons vu dans les chapitres consacrés à la gestion de l'écran et aux caractères utilisateur, l'écran est divisé en 40 colonnes et 25 lignes, soit 1 000 cases « caractères ». D'autre part, un caractère est une matrice de 8 × 8 = 64 points.

Chaque point représente la forme ou le fond d'un caractère ou d'un dessin et les points sont regroupés sur chaque ligne par paquets de huit, appelés GROUPE DE POINTS LIGNE (GPL).

Exemple : pour X = Ø à X = 7 se trouve le 1er GPL, de X = 8 à X = 15 le second...

Dans un groupe de points ligne, chaque point fait partie, ou de la forme, ou du fond, et l'on ne pourra donc définir dans un GPL que deux couleurs, la couleur de forme et la couleur de fond.

La fenêtre écran dans laquelle nous travaillons est donc constituée de 4Ø × 2ØØ = 8ØØØ GPL, chacun d'eux pouvant être numéroté. Le GPL situé en haut et à gauche (Y = Ø, X = Ø à 7) sera le GPL N° Ø ; puis, pour X = 8 à 15, le GPL 1, ..., pour Y = 1 et X = Ø à 7 nous trouverons le GPL N° 4Ø,... et ainsi de suite jusqu'au GPL n° 7999 en bas et à droite de la fenêtre de travail.

Pour définir complètement un GPL, il faut tout d'abord indiquer si un point appartient à la forme ou au fond du dessin ; comme un GPL contient 8 points, il faut donc un octet pour le représenter (fig. XIII-1).

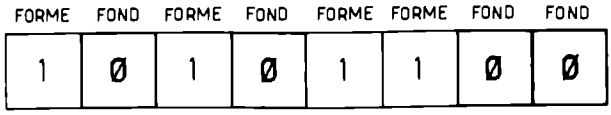


Fig. XIII-1. – Représentation binaire d'un GPL.

– Lorsqu'un bit de cet octet est à Ø, il appartient au fond.
– Lorsqu'un bit de cet octet est à 1, il appartient à la forme.

Pour représenter la totalité de la fenêtre, il faudra donc 8 K-octets de RAM.

Pour obtenir la couleur d'un point, on utilise les trois couleurs de base : bleu, vert et rouge. Pour définir une couleur, il faudra donc trois bits représentant :

– l'absence de la couleur de référence si un bit est à zéro,
– la présence de la couleur de référence si ce bit est à 1.

On obtient alors les huit couleurs du tableau XIII-1.

Pour chaque GPL, on peut définir la couleur des points appartenant à la forme, et la couleur de ceux appartenant au fond. On peut donc définir deux couleurs par GPL, ce qui nécessitera six bits. Il faudra donc une mémoire couleur de 8 K × 6 bits.

Bleu	Vert	Rouge		
Ø	Ø	Ø	Noir	Ø
Ø	Ø	1	Rouge	1
Ø	1	Ø	Vert	2
Ø	1	1	Jaune	3
1	Ø	Ø	Bleu	4
1	Ø	1	Magenta	5
1	1	Ø	Cyan	6
1	1	1	Blanc	7

Tableau XIII-1. – Synthèse des teintes.

En mode graphique les couleurs de fond doivent être complémentaires, c'est-à-dire :

NOIR	= – 1	BLEU	= – 5
ROUGE	= – 2	MAGENTA	= – 6
VERT	= – 3	CYAN	= – 7
JAUNE	= – 4	BLANC	= – 8

Chaque GPL a donc besoin de deux fois 8 K RAM pour définir les points et leur couleur. Deux blocs mémoire RAM sont placés aux mêmes adresses, chaque adresse correspondant au numéro d'ordre du GPL, un groupe mémorisant les points et l'autre les couleurs (voir fig. XIII-2).

La sélection d'un bloc mémoire parmi les deux se fait par la mise à Ø ou à 1 du bit CØ du PORT C du circuit 6846, se trouvant à l'adresse &HE7C3.

CØ à Ø sélectionne la mémoire de « couleur ».
CØ à 1 sélectionne la mémoire de « forme ».
(Voir le programme « dessin sur l'écran ».)

La mémoire écran se trouve donc de l'adresse &H4ØØØ à l'adresse &H5F3F.

La couleur du cadre est définie par les bits C4, C5, C6 du port C du 6846 (&HE7C3).

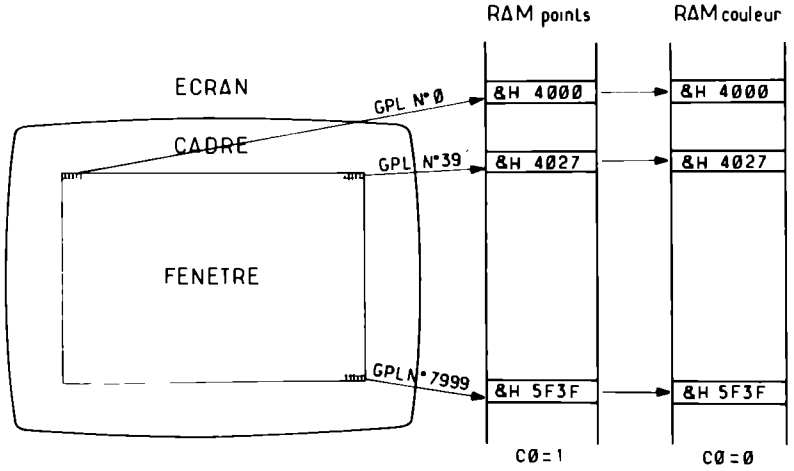


Fig. XIII-2. – Mémoire d'écran.

BOX (X_1, Y_1) – (X_2, Y_2) « Car », Attribut

Cette instruction permet le tracé des contours d'une boîte dont les coordonnées des sommets X_1, Y_1 et X_2, Y_2 correspondent à une diagonale.

Le contour de cette boîte peut être tracé à l'aide d'un caractère placé entre « ». Dans ce cas, l'attribut définit en trois valeurs (voir COLOR), la couleur des points, la couleur du fond, et l'inversion vidéo (couleurs de Ø à 7).

Le contour de la boîte peut être tracé en mode graphique. Dans ce cas, il n'y a ni caractère ni guillemets, et l'attribut ne définit que la couleur de ce contour-points à l'aide d'une valeur :

- de – 8 à – 1 si c'est un contour fond,
- de Ø à 7 si c'est un contour forme.

BOXF (X_1, Y_1) – (X_2, Y_2) « Car », Attribut

Cette instruction suit les mêmes règles que la précédente, mais dessine une boîte pleine.

LINE (X_1, Y_1) – (X_2, Y_2) « Car », Attribut

Cette instruction suit les mêmes règles que la précédente, mais tire une droite entre (X_1, Y_1) et (X_2, Y_2)

Si, dans ces trois instructions, (X_1, Y_1) est omis, le tracé se fait depuis le point courant, c'est-à-dire depuis (X_2, Y_2) de la dernière instruction de tracé utilisée.

PSET (X, Y) « Car », Attribut

Cette instruction dessine sur l'écran un point ou un caractère dont les couleurs précisées dans Attribut suivent les mêmes règles que précédemment.

POINT (X, Y)

Cette instruction renvoie la couleur (de – 8 à + 7) du point de coordonnées X, Y.

COLOR P, F, V

Cette instruction définit les couleurs points P et fond F ainsi que l'insertion vidéo si V est à 1.

SCREEN P, F, C, V

Comme la précédente, cette instruction définit les points P et fond F, l'inversion vidéo V, mais aussi la couleur du cadre C (de Ø à 7).

CHAPITRE XV **Applications**

Tracé de la courbe de réponse en fréquence d'un filtre R-C

Nous avons vu, au chapitre concernant les variables, que la fréquence de coupure du filtre R-C était $1/2\pi R_2 \cdot C$ et le gain en dB : $G = -4{,}34 * LN[1 + (R_2 * C * 2 * \pi * f \uparrow 2]$.

Dans une première partie du programme, on calcule la valeur de f et on affiche le tableau des résultats G (dB) = f pour 15 valeurs de fréquences.

Essayons maintenant de tracer la courbe du gain en fonction de f en coordonnées semi-logarithmiques de 10 Hz à 500 kHz.

Nous disposons de 320 points horizontaux ; or la longueur d'une décade 10-100, 100-1 000 est constante et, de 100 kHz à 500 kHz, on occupe $\log_{10} 5 = 0{,}7$ fois cette longueur. La longueur en points écran d'une décade est donc 320/4,7 = 85 points maximum. On choisira une longueur de 64 points.

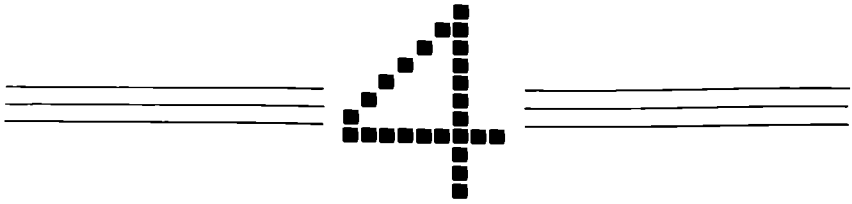
On dispose de 200 points en ordonnée. En prenant 1 dB = 3 points et en traçant la courbe jusqu'à – 40 dB, il suffit de 120 points, ce qui laisse suffisamment de place pour le titre, les axes et les valeurs du gain.

On en déduit donc le programme de la page suivante.

```
10 '              **************************
20 '              *                        *
30 '              * REPONSE D'UN FILTRE *
40 '              *                        *
50 '              **************************
60 '
70
80 '
82 '
84 '-------------Calcul du gain--------
85 '
88 '
90 A$="###.##^^^^":B$="##":C$="##^^^^"
100 DIMC(15),F(15):E=4.3429419:P=3.14159
27:R2=1E5
110 INPUT"Donner la valeur de C :";C
120 F0=1/(2*P*R2*C):CLS:PRINT"F0 = ";INT
(F0);"Hz"
130 FOR I=1 TO15
140     READ F(I)
150     G(I)=-E*LOG(1+(R2*C*2*P*F(I))^2)
160     PRINT"F(";:PRINTUSINGB$;I;:PRINT"
)=";:PRINTUSINGA$;F(I);:PRINT"     G(";:P
RINTUSINGB$;I;:PRINT")=";:PRINTUSINGA$;G
(I)
170     NEXTI
180 DATA10,20,50,100,200,500,1E3,2E3,5E3
,1E4,2E4,5E4,1E5,2E5,5E5
182 '
184 '
186 '------------Trace ou fin ?----------
188 '
190 PRINT"Voulez-vous la trace de la cou
rbe ?"
200 R$=INKEY$:IFR$=""GOTO200ELSEIFR$="N"
THENENDELSEIFR$="O"GOTO210ELSEGOTO200
202 '
204 '
```

```
200 '---------Trace des axes----------------
208 '
210 CLS:SCREEN7,0,0
220 LINE(24,24)-(24,193):LINE(22,193)-(2
4,198):LINE(26,193)-(24,198)
230 LINE(20,30)-(313,30):LINE(313,28)-(3
19,33):LINE(313,32)-(316,30)
240 LOCATE2,2:PRINT"10"SPC(6)"100"SPC(5)
"1K"SPC(6)"10K"SPC(4)"100K"SPC(3)"F"
250 LOCATE0,4:PRINT"0":LOCATE0,8:PRINT"-
10":LOCATE0,12:PRINT"-20":LOCATE0,16:PRI
NT"30":LOCATE0,20:PRINT"-40":LOCATE1,24
260 '
265 '
264 '---------Trace de    courbe--------
265 '
268 '
270 PSET(24,24):COLOR1:R=1
280 FORN=12TO300STEP12
290     R=N*R:X=INT(28*LOG(R)-40)
300     G=INT(D*LOG(1+(D2*C*2*P*F)^2))
310     Y=24+3*G
320     IFY>=193THENEND
330     LINE-(X,Y)
340     NEXTN
350 R=10*R:IFR>100000THENENDELSEGOTO280
```

Page blanche



Le microprocesseur 68Ø9

CHAPITRE XVI **Caractéristiques générales**

Le 68Ø9 est un microprocesseur 8 bits, c'est-à-dire qu'il est capable de traiter les informations (les données) 8 par 8 (octet par octet). L'ensemble de ces huit informations binaires circulera dans tout le micro-ordinateur, le long de ce qu'on appellera le BUS de DONNEES.

La conception tant matérielle que logicielle du 68Ø9 fait de lui le microprocesseur idéal pour les techniques de programmation moderne (banalisation de l'instruction, réentrance des sous-programmes, programmation modulaire) et l'implantation de langages évolués de haut-niveau.

Bien que possédant moins d'instructions que le 68ØØ (59 au lieu de 72), l'augmentation très importante de ses modes d'adressage et du nombre de registres, ainsi que la possibilité d'appliquer pratiquement toute instruction à n'importe quel registre en font un des microprocesseurs les plus puissants actuellement sur le marché, puisqu'il permet 1464 instructions différentes (voir annexe 10).

Pour l'utilisateur du TO 7, c'est principalement cette étude du logiciel qui est intéressante, car l'implantation matérielle est déjà réalisée, et seule nous intéresse la possibilité de programmer en assembleur.

CHAPITRE XVII **Structure interne**

VOIR ANNEXE 1

1. LES ACCUMULATEURS A, B et D

Le 68Ø9 contient deux registres de travail de 8 bits appelés Accumulateurs A et B. Ces registres peuvent stocker chacun un octet et de nombreuses instructions leur sont applicables.

Par l'intermédiaire du bus interne du 68Ø9, ils sont donc reliés au bus de données. Ce dernier étant bidirectionnel, on pourra donc :

- faire entrer un octet dans un accumulateur,
- faire sortir un octet d'un accumulateur,
- manipuler un octet d'accumulateur, tout en restant à l'intérieur du 68Ø9.

Une particularité intéressante du 68Ø9 est la possibilité logicielle de considérer que ces deux accumulateurs n'en font plus qu'un, appelé accumulateur D, formé donc de 16 bits (2 octets) et obtenu en juxtaposant A et B, de telle façon que A contienne l'octet de poids fort et B l'octet de poids faible de D (voir fig. XVII-1).

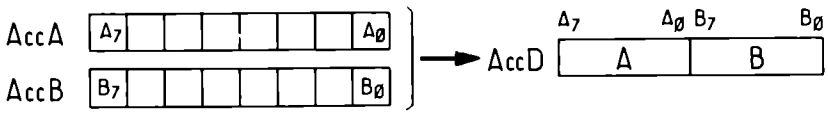


Fig. XVII-1. – L'accumulateur D.

Des instructions existeront donc avec le 68Ø9 qui permettront de manipuler les données par paquets de 16 bits bien que le bus de données soit un bus 8 bits. C'est cette caractéristique qui fait souvent dire que le *68Ø9 est un microprocesseur pseudo 16 bits.*

2. LES REGISTRES U, S, X et Y

Le microprocesseur 68ØØ contenait un registre d'index X et un pointeur de pile S. Le 68Ø9 a fait plus que doubler ces caractéristi-

ques. En effet, on peut y trouver deux registres d'index X et Y et deux pointeurs de pile S et U mais, en outre, la totale symétrie des instructions fait que les premiers peuvent jouer le rôle des seconds et vice-versa et que l'on peut échanger et transférer les contenus de tous ces registres entre eux, puisque ce sont tous en effet des registres 16 bits.

Etant reliés au bus de données on peut donc :

- faire entrer deux octets dans un registre, en deux fois,
- faire sortir deux octets d'un registre soit vers le bus de données, en deux fois, soit d'un seul coup vers le bus d'adresses,
- manipuler le contenu d'un registre de la façon interne au 68Ø9.

En particulier, grâce à la possibilité de concaténer A et B, on pourra échanger les contenus de tous les registres 16 bits entre eux, y compris l'accumulateur D et *même le compteur programme PC.*

3. LE REGISTRE COMPTEUR PROGRAMME

Ce compteur 16 bits a pour rôle de faire passer sur le bus d'adresse l'adresse de la mémoire, du circuit d'interface,... auquel correspond le traitement de l'instruction en cours.

Grâce aux énormes possibilités logicielles du 68Ø9, on pourra considérer ce compteur comme un registre 16 bits ordinaire dont on pourra par exemple échanger le contenu avec D, X, Y, S ou U.

4. LE REGISTRE DE PAGE DP

Ce registre 8 bits est en permanence relié à la moitié haute du bus d'adresse. Il contient donc l'octet de poids fort de l'adresse, ce qui va permettre en mode d'adressage direct d'atteindre l'une quelconque des 65 535 adresses du TO 7.

Alors qu'avec le 68ØØ on ne pouvait travailler en mode direct que sur la page Ø, c'est-à-dire des adresses &HØØØØ à &HØØFF, on peut cette fois aller de &HØØFF à &HFFFF en modifiant le contenu de DP.

1re remarque : une fois DP chargé par un octet, le mode d'adressage direct ne s'étend que sur 256 adresses de (DP)ØØ à (DP)FF.

2e remarque : pour conserver sa compatibilité ascendante de logiciel avec le 68ØØ, le 68Ø9 remet à zéro le registre DP à la mise sous tension.

3e remarque : il n'y a pas d'instruction qui permette de charger directement ce registre DP avec l'octet choisi. Il faut donc charger, par exemple, un accumulateur puis échanger son contenu avec DP.

5. LE REGISTRE DE CODES CONDITION CCR

Ce registre 8 bits donne des indications sur le résultat de la dernière opération réalisée par le 68Ø9 et permet également de masquer des interruptions ou de connaître l'état du 68Ø9 lors d'une opération d'empilage ou de dépilage.

Chaque bit de ce registre est indépendant des autres et contient une information à lui seul (voir fig. XVII-2).

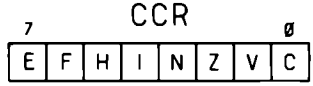


Fig. XVII-2. – Le registre de codes condition.

Le BIT Ø (C)

Ce bit indique, quand il passe à 1, que la dernière opération a donné lieu à une retenue.

Lors d'une soustraction (en complément à 2), il représente donc le complément de la retenue.

Le BIT 1 (V)

Ce bit indique, quand il passe à 1, que la dernière opération (en complément à deux signé) a donné lieu à un débordement, c'est-à-dire que la retenue du bit de poids le plus fort n'est pas égale à celle du bit immédiatement inférieur.

Exemple :

On veut additionner les valeurs décimales – 63 et – 110.
+ 63 s'écrit en binaire : Ø Ø 1 1 1 1 1 1

Pour obtenir – 63, on complémente à 1 puis on ajoute 1 (complément à 2) :

```
   1 1 Ø Ø Ø Ø Ø Ø
+                1
-----------------
   1 1 Ø Ø Ø Ø Ø 1
```

Le bit de poids le plus fort représente le signe ,Ø pour + et 1 pour – et permet donc de savoir si les bits qui le précèdent représentent la valeur immédiate du nombre où son complément à deux.

De la même façon :

	+	1	1	Ø	s'écrit	Ø	1	1	Ø	1	1	1	Ø
et	–	1	1	Ø		1	Ø	Ø	1	Ø	Ø	1	Ø

Pour reconvertir un nombre binaire en complément à deux signé, on ne tient compte que des 7 bits de poids faibles dont on prend la valeur décimale (exemple ci-dessus, 18) et l'on retranche cette valeur de 128 (exemple ci-dessus, 11Ø), puis l'on replace le signe moins.

Additionnons – 63 et – 11Ø :

		Ø						
		1	1	Ø	Ø	Ø	Ø	Ø	1
	+	1	Ø	Ø	1	Ø	Ø	1	Ø
Retenue C →	[1]	Ø	1	Ø	1	Ø	Ø	1	1

La retenue entre les bits 6 et 7 est nulle, alors que celle portant sur les bits 7 est égale à 1. Le bit de retenue C passe donc à 1 ainsi que le bit de débordement V. On en déduit donc que le résultat est négatif et dépasse la valeur maximum représentable sur 8 bits, soit – 128.

Le BIT 2 (Z)

Ce bit indique, quand il passe à 1, que le résultat de la dernière instruction est nul.

Le BIT 3 (N)

Ce bit indique, quand il passe à 1, que le résultat de la dernière instruction est négatif.

Le BIT 4 (I)

Ce bit, lorsqu'il est à 1, empêche la prise en compte des interruptions IRQ.

Le BIT 5 (H)

Ce bit indique, quand il passe à 1, que la dernière opération a donné lieu à une retenue des bits 3 sur les bits 4, c'est-à-dire une retenue du premier demi-octet sur le deuxième (poids forts).

On l'appelle bit de demi-retenue, et il est surtout utilisé dans les opérations portant sur les valeurs décimales.

Le BIT 6 (F)

Ce bit, lorsqu'il est à 1, masque les interruptions du type FIRQ en n'autorisant que les interruptions plus prioritaires.

Le BIT 7 (E)

Ce bit indique, quand il est à 1, que tous les registres du 68Ø9 ont été sauvegardés dans la pile.

Tous ces bits permettront de faire des tests, suite à une opération, afin d'orienter convenablement la suite du programme vers l'une ou l'autre partie suivant qu'un bit sera à Ø ou à 1.

6. NOTION DE PILE

Lorsque l'on veut quitter un programme pour aller vers un sous-programme, il est nécessaire de sauvegarder en mémoire l'état des divers registres du microprocesseur, avant le départ en sous-programme, afin qu'au retour on puisse récupérer toutes ces valeurs.

Les cases mémoires sont toutes adjacentes, ce qui donne l'impression d'une pile, un peu comme une pile d'assiettes qu'on remplirait par le bas (adresse la plus forte) en remontant vers le haut (adresses de plus en plus faibles).

L'ordre d'empilement est toujours le même : on commence par ranger PC au fond de la pile, puis U juste au-dessus, puis dans l'ordre Y, X, DP, B, A et CCR (voir fig. XVII-3). Dans cet exemple, on travaille avec la pile S, et à chaque instant le registre 16 bits S, appelé POINTEUR de PILE, contient l'adresse de la dernière case mémoire utilisée. Il pointe donc le haut de la pile.

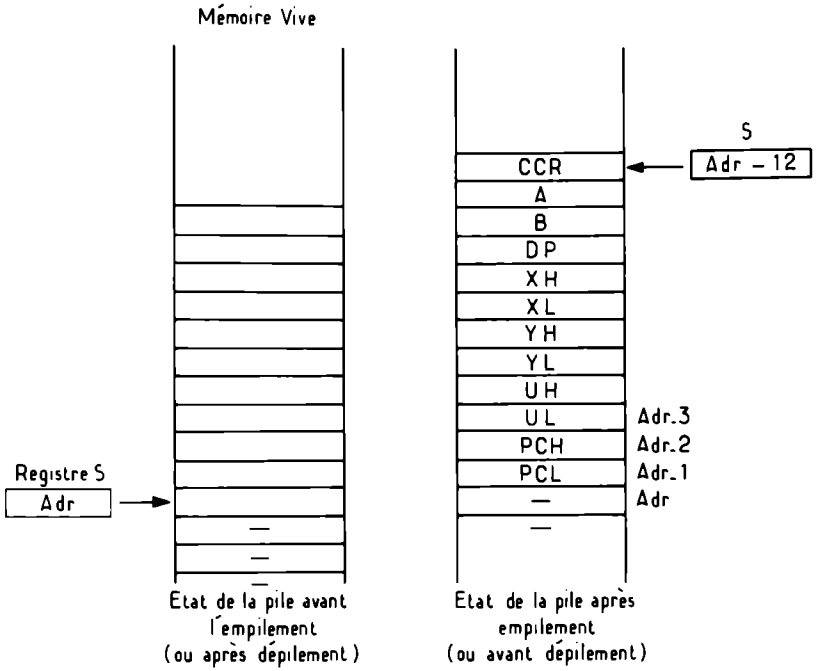


Fig. XVII-3. – Manipulation de la pile.

Le dépilement se fait exactement dans l'ordre inverse car, comme pour une pile d'assiettes, on prend d'abord celle du dessus, puis celle immédiatement en dessous, etc.

La mémoire étant une mémoire 8 bits, on ne peut y ranger un registre 16 bits d'un seul bloc. On commence donc toujours par l'octet de poids faible (LOW) exemple PCL, puis par celui de poids fort (HIGH), exemple PCH. C'est ce qui explique qu'une sauvegarde complète des registres du 68Ø9 nécessite 12 cases mémoires.

Un empilage complet fait passer le BIT E à 1.

La pile S est en général la pile système, alors que la pile U est réservée à l'utilisateur. N'oublions pas que ces deux registres peuvent servir également pour l'adressage indexé.

CHAPITRE XVIII **Les modes d'adressage**

1. ADRESSAGE INHERENT

Dans ce mode d'adressage, il n'y a aucun échange avec l'extérieur du microprocesseur. On l'appelle également adressage implicite, en ce sens que justement il n'y a pas à fournir d'adresse de départ ou de destination des données. Les instructions de ce type n'utilisent qu'un ou deux octets comme code machine hexadécimal. Elles sont les plus rapides à exécuter.

Exemple : l'instruction COMA, qui prend le contenu de l'accumulateur A et le complémente à 1 pour le remettre dans A, a pour code machine &H43 et nécessite deux cycles pour son exécution.

Cette opération est décrite comme suit : $\overline{A} \rightarrow A$.

A la suite de celle-ci, les bits N et Z prennent la valeur correspondant à A :

- si A = ØØØØØØØØ Z = 1 sinon Z = Ø
- si le bit de poids fort de A est 1 → N = 1 sinon N = Ø

Le bit V est forcé à Ø
Le bit C est forcé à 1
Le bit H est inchangé.

On vérifiera ce fonctionnement en se reportant à l'annexe « Jeu d'instructions du 68Ø9 ».

Exemple : L'instruction EXG R1,R2 qui permet l'échange des contenus des registres R1 et R2 a un code machine sur deux octets et nécessite huit cycles machine.

Le premier octet, qui indique que l'opération est un échange, a pour code machine &H1E.

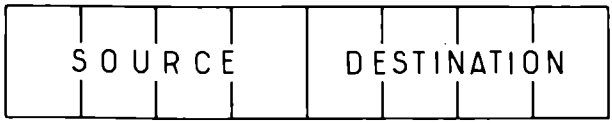


Fig. XVIII-1. – POST-BYTE des instructions TFR/EXG.

Le deuxième octet contient les codes du registre SOURCE et du registre DESTINATION (appelés comme ceci pour l'opération de transfert TFR, car pour EXG on met R1 dans R2 et R2 dans R1).

Les codes des registres sont les suivants :

0 0 0 0 = D (A:B)	1 0 0 0 = A
0 0 0 1 = X	1 0 0 1 = B
0 0 1 0 = Y	1 0 1 0 = CCR
0 0 1 1 = U	1 0 1 1 = DP
0 1 0 0 = S	
0 1 0 1 = PC	

Si l'on veut échanger les contenus de X et Y, on peut écrire :

1E 12, c'est-à-dire EXG X,Y

ou : 1E 21, c'est-à-dire EXG Y,X

En effet le code du registre source occupe le 1/2 octet de poids fort et le code du registre destination le 1/2 octet de poids faible (voir fig. XVIII-1).

Cet octet supplémentaire est appelé POST-BYTE. On remarquera que, bien évidemment, si on peut échanger entre eux deux registres 16 bits ou deux registres 8 bits il est par contre impossible d'échanger un registre 16 bits avec un registre 8 bits.

2. *ADRESSAGE IMMEDIAT*

Dans ce mode d'adressage, la valeur à traiter suit immédiatement le code opération.

Donc, si l'opération concerne un registre 8 bits, il faudra deux octets de code.

Si l'opération concerne un registre 16 bits, il faudra trois octets de code.

Ce mode d'adressage reste encore parmi les plus rapides (de 2 à 4 cycles).

Exemples : On peut charger un accumulateur, A par exemple, avec la valeur numérique &H1A en écrivant :

LDA # $ 1A	86 1A	(Voir fig. XVIII-2)
Mnémonique Assembleur	Code Hexadécimal	

En langage d'assemblage, le # symbolise l'adressage immédiat, le $ symbolisant toujours une valeur hexadécimale.

- On peut ajouter une valeur à l'accumulateur D :
  ADDD # $ 1Ø01 → C3 1Ø01

- On peut comparer un registre d'index à une valeur numérique :
  CMPX # $ 1EF3 → 8C 1E F3

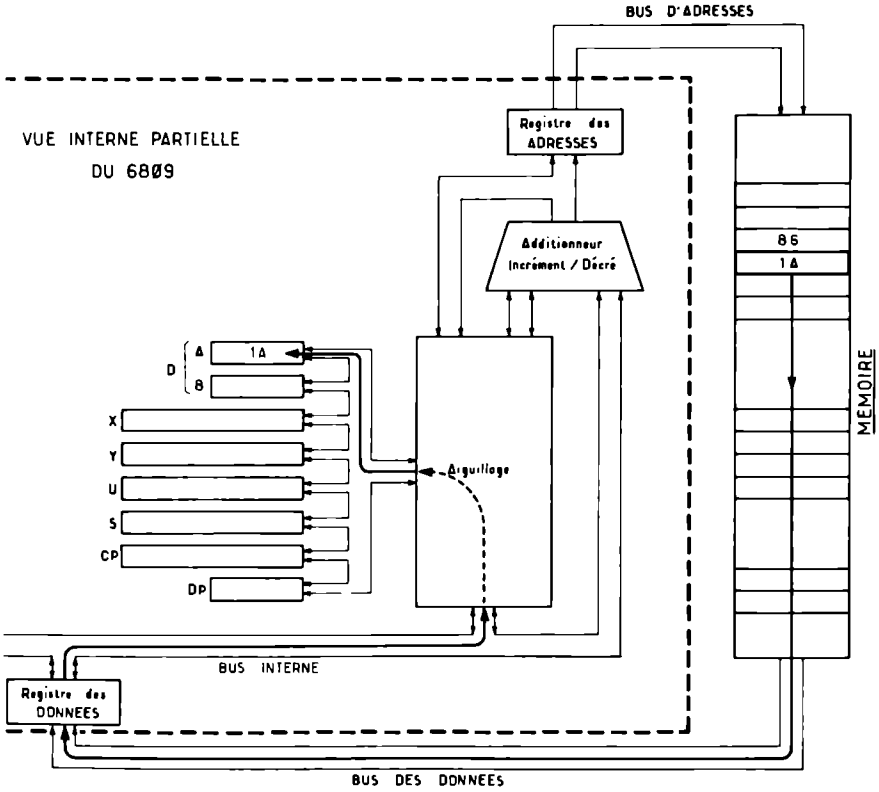


Fig. XVIII-2. – Adressage immédiat.

3. ADRESSAGE ETENDU

Dans ce mode d'adressage, la valeur numérique à traiter se trouve rangée à une *adresse* connue, et c'est cette adresse que l'on indique *sur deux octets* à la suite du code opération.

Dans presque tous les cas, il faudra donc trois octets pour définir toute l'instruction. Seules exceptions, les quelques instructions dont le code opération tient sur deux octets.

La durée de traitement est de cinq à huit cycles.

Exemples :

– On veut charger l'accumulateur B avec une donnée qui se trouve à l'adresse $EFØØ :

LDB $ EFØØ → F6 EF ØØ

On remarquera la disparition du # de la syntaxe Assembleur.

– On veut charger le pointeur U avec une donnée qui se trouve à l'adresse $8Ø1C. En fait, le pointeur U étant un registre 16 bits ne peut être chargé que par des informations de 16 bits, ce qui nécessite deux adresses mémoire consécutives, c'est-à-dire, dans l'exemple présent, $8Ø1C et $8Ø1D. Mais on n'indique dans la syntaxe que l'adresse la plus basse (qui contient l'octet de poids fort de la donnée, alors que Adresse + 1 contient l'octet de poids faible).

LDU $ 8Ø1C → FE 8Ø 1C

(Voir fig. XVIII-3)

– Pour charger le pointeur S avec la même donnée, on écrira :

LDS $ 8Ø1C → 1Ø CE 8Ø 1C

Code op.

Mais le code opération de cette instruction occupe deux octets.

4. ADRESSAGE DIRECT

Dans ce mode d'adressage, la valeur numérique à traiter se trouve à une adresse connue et on indique, derrière le code opération, l'octet de poids faible de cette adresse. Ceci suppose que le registre de page ait été préalablement chargé avec l'octet de poids fort de l'adresse.

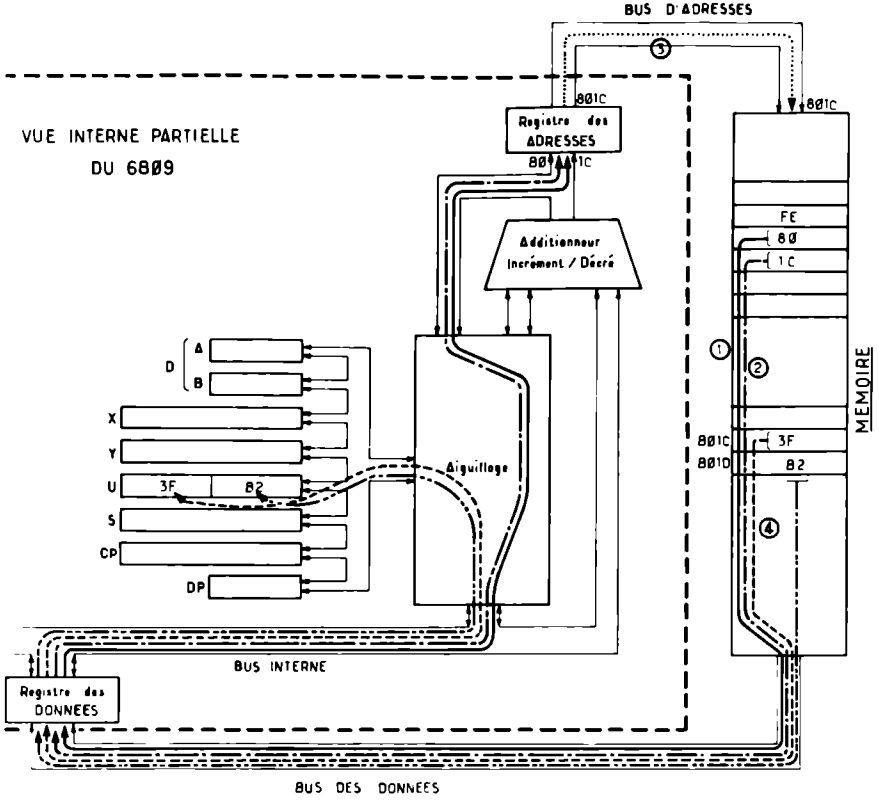


Fig. XVIII-3. – Adressage étendu.

Reprenons l'exemple précédent ; on écrira :

LDU $ 1C → DE 1C

(Voir fig. XVIII-4)

Le registre DP contient $8Ø.

L'intérêt de ce mode d'adressage est qu'il est à la fois plus rapide et plus économe en mémoire que l'adressage étendu, à condition toutefois que l'on ait à manipuler plusieurs données se trouvant à la même page car, sinon, le temps passé à charger et recharger DP annule les bénéfices de l'opération.

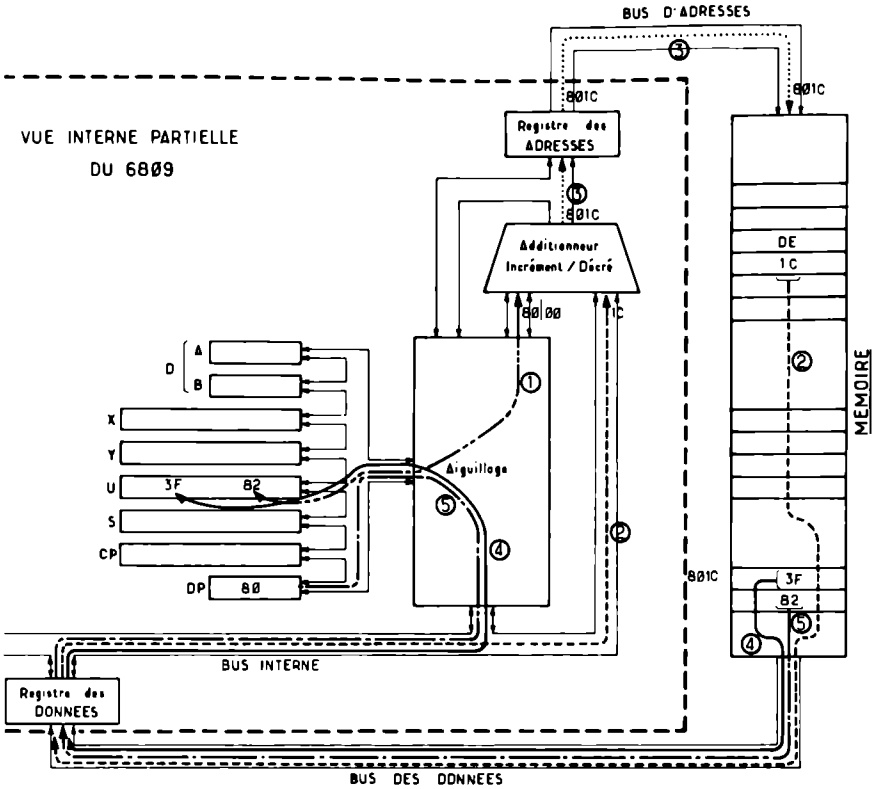


Fig. XVIII-4. – Adressage direct.

5. ADRESSAGE ETENDU INDIRECT

Dans ce mode d'adressage, la donnée est à une adresse que l'on ne connaît pas directement. Mais on sait où est rangée cette adresse, c'est-à-dire que l'on connaît l'adresse de l'adresse.

Exemple : On veut comparer le contenu de l'accumulateur A avec une valeur rangée à l'adresse Adr. La valeur de cette adresse est elle-même rangée en &H6700, par exemple.

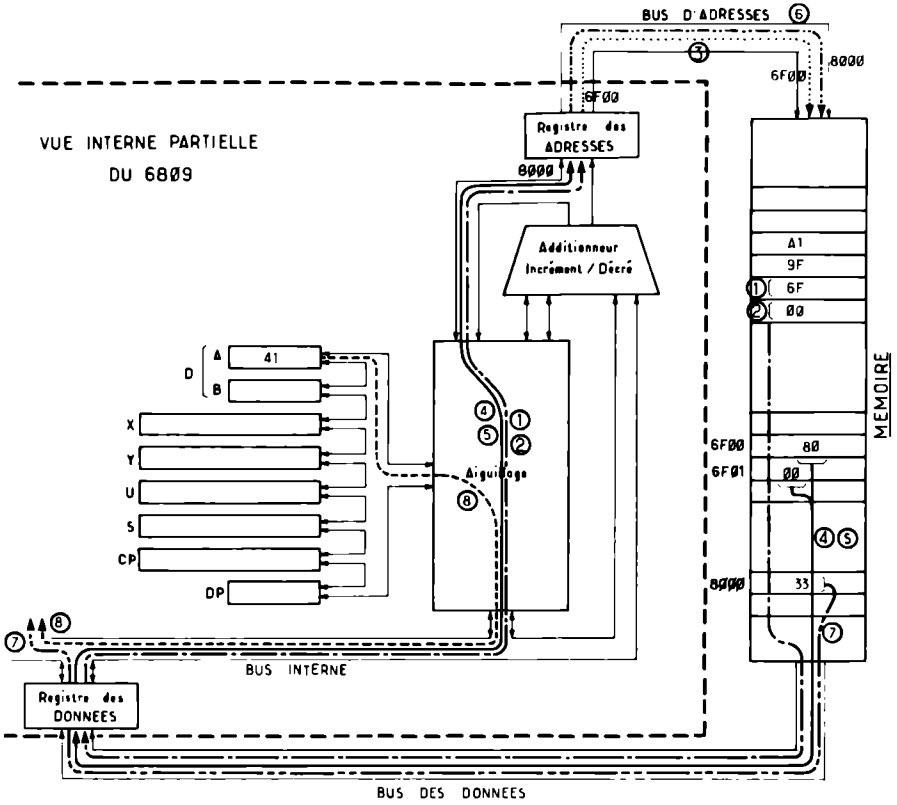


Fig. XVIII-5. – Adressage étendu indirect.

Le code opération est celui du mode indexé suivi de &H9F, soit deux octets dans le cas général (parfois, lorsque l'opération portera sur des pointeurs, un pré-octet viendra s'ajouter au code opération).

On écrira donc ici :

CMPA [$6700] → A1 9F 6F 00

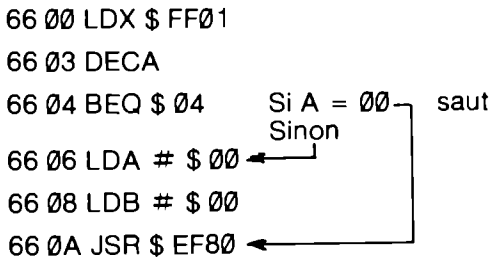
(Voir fig. XVIII-5)

La procédure sera alors celle de la figure. Les deux crochets indiquent qu'il s'agit d'un adressage indirect.

6. ADRESSAGE RELATIF COURT

Ce mode d'adressage est utilisé uniquement dans les opérations de branchement. Un branchement est réalisé à la suite d'un test, si ce dernier est vrai. Par exemple, si on souhaite sauter quatre lignes de programme si la dernière opération a donné un résultat nul, on dira « Branchement si Egal (à zéro) » soit en anglais « Branch if EQUAL » → le mnémonique BEQ.

Donc on obtient BEQ$Ø4. Cette instruction provoque un saut des quatre lignes qui suivent et le programme ne vient donc lire que la cinquième ligne.



Dans ce mode d'adressage, on ne fournit donc pas l'adresse réelle de destination. Celle-ci est en effet calculée par rapport à la valeur du compteur programme. Si le test conditionnant le branchement est vrai, alors la valeur qui suit le code opération est ajoutée au contenu du compteur programme pour obtenir l'adresse de destination.

Lorsque le microprocesseur a terminé de lire les deux octets de l'instruction de branchement, le compteur programme est déjà pointé sur la case mémoire suivante (par exemple ici &H66Ø6). On en déduit donc la formule de calcul de l'adresse relative :

Adr. Relative = Adr. Destination – (Adr. Brancht + 2)

Le branchement pouvant se faire aussi bien en avant (positif) qu'en arrière (négatif), la valeur relative est donc signée, le bit de signe étant le bit de poids fort.

On ne peut donc faire un saut que de + 127 (&H7F) en avant, et de – 128 (&HØØ) en arrière. La valeur hexadécimale de branchement en arrière doit donc être donnée en mode complément à 2.

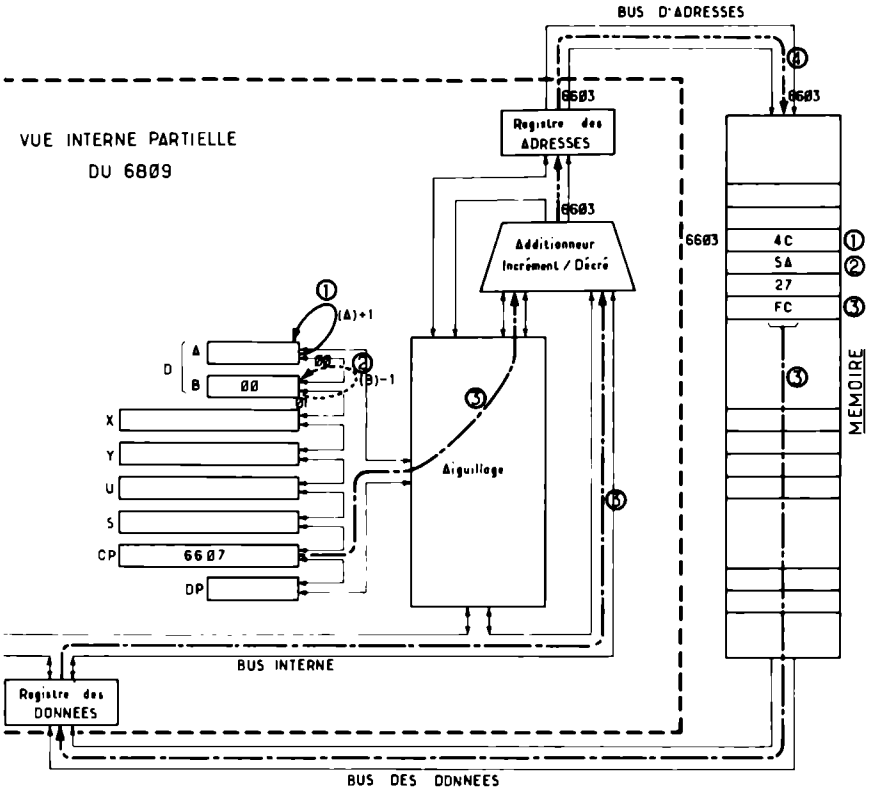


Fig. XVIII-6. – Adressage relatif court.

Exemple : 66 00 LDX $ F 000

Adr	66 03 INCA	4C
	66 04 DECB	5A
	66 05 BEQ [Adr]	27 FC
	66 07 JSR $ EF 08	

$Adr = COM_2 (| \&H6603 - \&H6607 |) = COM_2(\&H04)$

$\&H04 = 00000100$

$COM_2 (\&H04) = 11111100 = \&HFC$

Donc on écrira en 6605 BEQ $ FC

7. ADRESSAGE RELATIF LONG

Ce mode d'adressage, de même principe que le précédent permet des sauts sur la totalité de la zone mémoire, c'est-à-dire de + 32767 (&H7FFF) en avant, et – 32768 (&H8ØØØ) en arrière.

Le code opération contient toujours le pré-octet &HlØ.

Exemple : BEQ $Ø4 peut aussi s'écrire
LBEQ $ØØØ4 → 1Ø27ØØØ4

8. ADRESSAGE INDEXE

Dans ce mode d'adressage, l'un des registres 16 bits X, Y, U, S, ou le compteur programme, sert d'INDEX pour le calcul de l'adresse effective de la valeur à traiter.

L'adresse de la valeur à traiter est calculée à partir de la valeur contenue dans l'INDEX de référence, choisi par l'utilisateur, et appelé BASE.

On distingue plusieurs possibilités d'indexation.

a) Adressage indexé à déplacement nul

Dans ce mode, la base sélectionnée contient l'adresse où se trouve la valeur à traiter.

Le code opération est suivi d'un post-octet de valeur binaire : 1BBØØ1ØØ (voir tableau ci-dessous).

Les deux octets B B définissent la base conformément au tableau :

B	B	Registre
Ø	Ø	X
Ø	1	Y
1	Ø	U
1	1	S

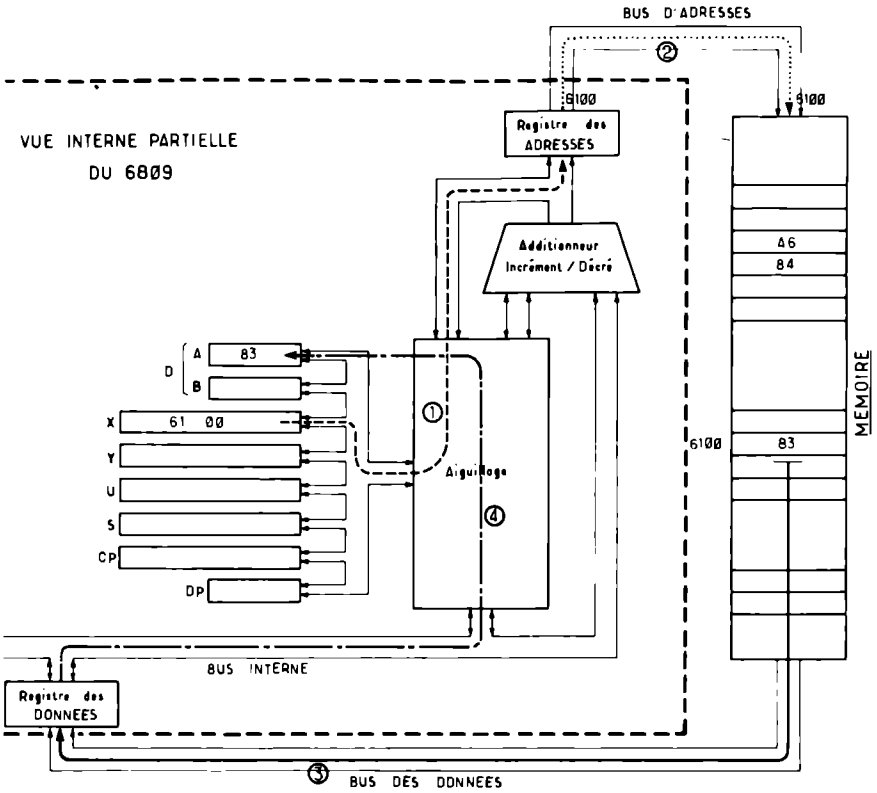


Fig. XVIII-7. – Adressage indexé à déplacement nul.

Exemple : On veut charger l'accumulateur A avec la valeur qui se trouve en 6100. Le registre X contient l'adresse &H6100.

X = &H6100

LDA 0,X → A6 84

(Voir fig. XVIII-7)

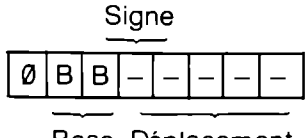
C'est le mode indexé le plus rapide.

b) Adressage indexé à déplacement constant

Dans ce mode d'adressage, un déplacement est ajouté à la base choisie pour obtenir l'adresse de la donnée. Ce déplacement peut être codé sur 5, 8 et 16 bits.

Dans chaque cas, le bit de poids fort est le bit de signe. Les valeurs négatives sont en mode complément à 2.

– **Déplacement codé sur 5 bits :** dans ce cas, on peut se déplacer par rapport à la base de – 16 à + 15. La valeur du déplacement est contenue dans le post-octet :

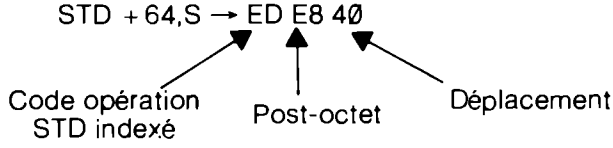


Pour comparer B à une valeur située à l'adresse (X)+2, on écrira : CMPB 2,X → E1 Ø2

Pour réaliser un OU logique entre A et une valeur située à l'adresse (U)–3, on écrira : ORA –3,U → AA 5D

– **Déplacement codé sur 8 bits :** dans ce cas, on peut se déplacer par rapport à la base de – 128 à + 127. La valeur du déplacement suit le post-octet. Ce dernier s'écrit : 1BBØ1ØØØ.

Pour stocker D à l'adresse (S)+64, on écrira :

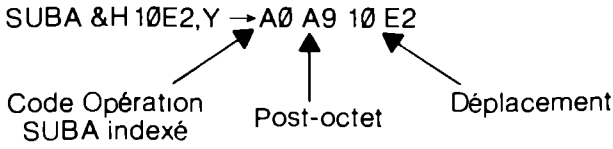


(Voir fig. XVIII-8)

– **Déplacement codé sur 16 bits :** dans ce cas, on peut se déplacer par rapport à la base de – 32768 à + 32767. La valeur du déplacement est codée sur deux octets qui suivent le post-octet.

Ce dernier s'écrit : 1BBØ1ØØ1.

Pour soustraire à A une valeur située à l'adresse (Y)–&H1ØE2, on écrira :



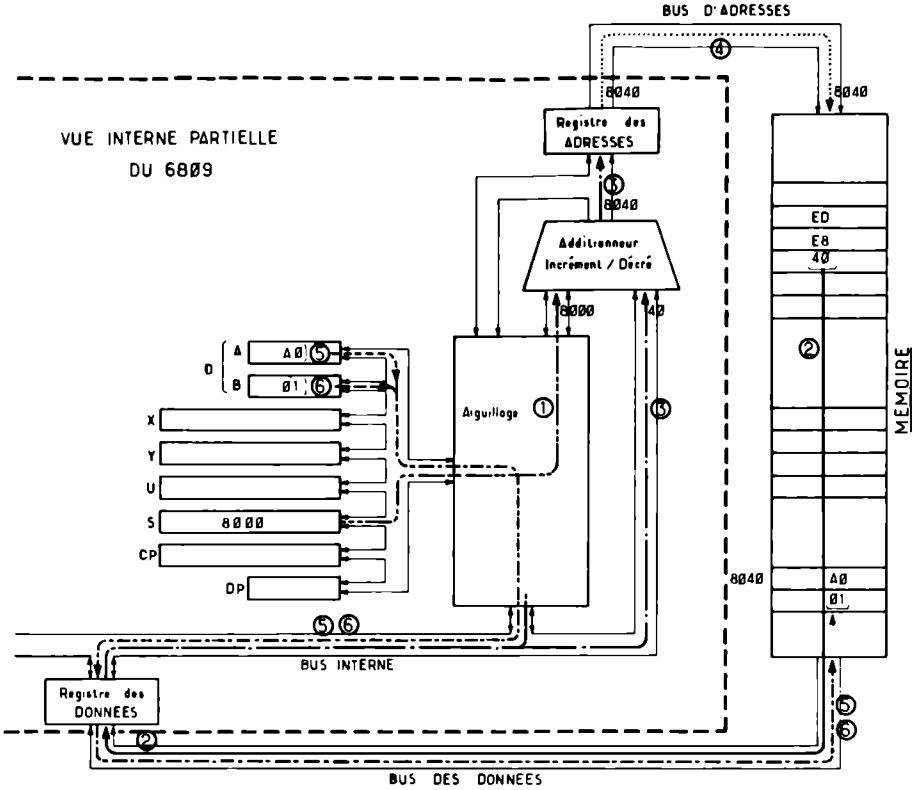


Fig. XVIII-8. – Adressage indexé – déplacement constant sur 8 bits.

c) Adressage indexé avec auto-incrémentation/décrémentation

Dans ce mode d'adressage, la base contient l'adresse de destination.

En auto-incrémentation, la valeur contenue dans la base est incrémentée de un ou de deux *après* avoir effectué l'opération.

En auto-décrémentation, la valeur contenue dans la base est décrémentée de un ou de deux *avant* d'avoir effectué l'opération.

Le post-octet a l'une des formes ci-dessous.

1 B B Ø Ø Ø Ø Ø	,BASE +
1 B B Ø Ø Ø Ø 1	,BASE+ +
1 B B Ø Ø Ø 1 Ø	,–BASE
1 B B Ø Ø Ø 1 1	,– – BASE

Ce mode d'adressage est particulièrement utile pour la gestion de tableaux de valeurs.

– Lorsque les valeurs de données ont 8 bits, on travaille en mode auto-incrémentation/décrémentation de un.

– Lorsque les valeurs des données ont 16 bits, on incrémente ou décrémente de deux.

On peut utiliser les registres X et Y comme des piles logicielles ainsi que U et S.

d) Adressage indexé – Déplacement accumulateur

Dans ce mode d'adressage, l'adresse effective est obtenue en ajoutant à la base le contenu de l'un quelconque des accumulateurs A, B ou D.

Le post-octet est alors le suivant :

1 B B Ø 1 Ø 1 1 pour Acc D
1 B B Ø Ø 1 1 Ø pour Acc A
1 B B Ø Ø 1 Ø 1 pour Acc B

Exemple : On veut charger l'accumulateur A avec la valeur qui se trouve à l'adresse obtenue en ajoutant le contenu de l'accumulateur B à celui du registre Y. On écrit donc :

LDA B,Y → A6 A5

On remarquera que, dans ce cas, l'accumulateur B (8 bits) vient s'ajouter à l'octet de poids faible du registre Y (16 bits).

(Voir fig. XVIII-9)

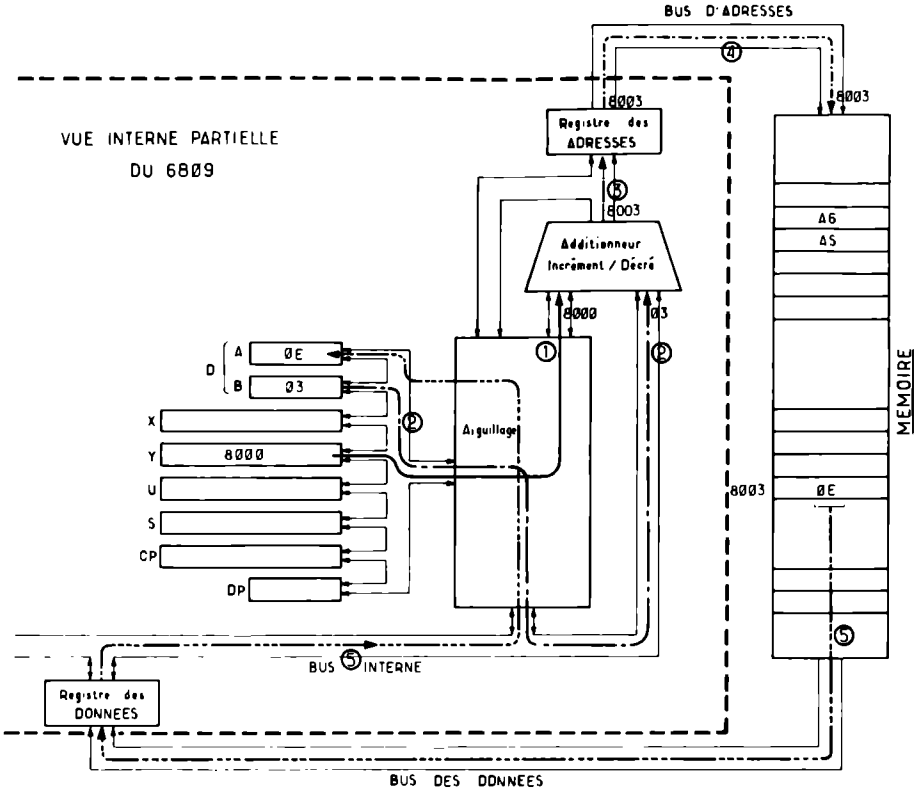


Fig. XVIII-9 – Adressage indexé. Déplacement accumulateur.

e) Adressage indexé – Base = compteur-programme

Dans ce mode d'adressage, le compteur programme est la base de calcul de l'adresse. Il s'agit donc en fait d'un adressage relatif qui peut avoir deux modes, long ou court.

– Déplacements sur 8 bits :

Dans ce cas, on peut se déplacer de – 128 à + 127 par rapport au compteur programme. L'octet de déplacement suit le post-octet qui s'écrit :

1 X X Ø 1 1 Ø Ø

Valeur quelconque

– Déplacement sur 16 bits :

Dans ce cas, on peut se déplacer de – 32768 à + 32767 par rapport au compteur-programme. Les deux octets du déplacement suivent le post-octet qui s'écrit :

1 X X Ø 1 1 Ø 1

Valeur quelconque

9. ADRESSAGE INDEXE-INDIRECT

Dans ce type d'adressage, la valeur calculée à partir de la base et du déplacement, contient cette fois l'adresse de l'adresse de la valeur à traiter.

Une indirection s'ajoute donc à l'indexation.

On obtient ce mode d'adressage en faisant passer à 1 le bit 4 du post-octet.

L'indirection n'est pas possible avec les modes indexés suivants :

– déplacement constant sur 5 bits
– auto-incrémentation/décrémentation de un.

Exemple : On veut charger A avec une valeur dont l'adresse se trouve à l'adresse obtenue en ajoutant &H1Ø au registre X. On écrit :

LDA [1Ø,X] → A6 98 1Ø.

(Voir fig. XVIII-10)

10. L'INSTRUCTION LEA (LOAD EFFECTIVE ADDRESS)

Cette instruction, qui ne s'utilise qu'en mode indexé, ne concerne que les registres d'index X, Y, U et S. Elle permet de charger, dans le registre choisi, une valeur calculée à partir de l'indexation, l'index pouvant être un autre registre.

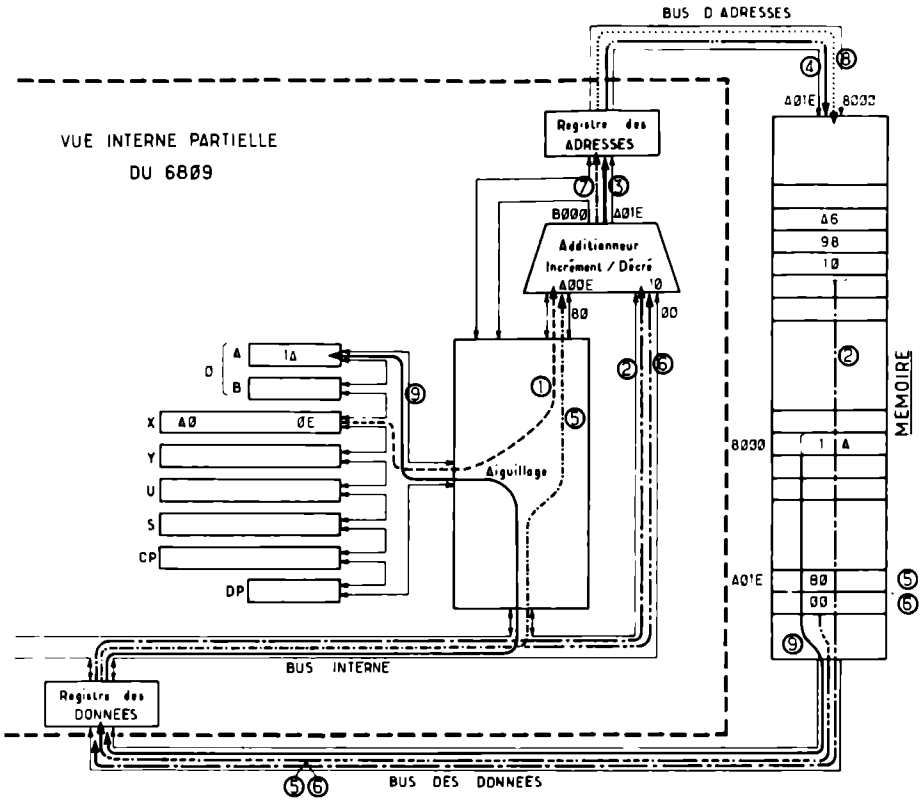


Fig. XVIII-10 – Adressage indexé indirect.

Exemple :

On peut écrire LEAY $ Ø8,X → 31 Ø8

Ce qui revient à mettre dans Y la valeur qui se trouve à l'adresse calculée en ajoutant Ø8 au contenu de X.

L'indirection est applicable à cette instruction.

Exemple :

Si l'on écrit LEAU [1F82,U] → 33 D9 1F 82.

On charge U avec la valeur dont l'adresse se trouve à l'adresse calculée en ajoutant 1F82 au contenu de U.

CHAPITRE XIX **Le jeu d'instructions du 6809**

Tableau XIX-1. – 8-Bit Accumulator and Memory Instructions.

Instruction	Description
ADCA, ADCB	Add memory to accumulator with carry
ADDA, ADDB	Add memory to accumulator
ANDA, ANDB	And memory with accumulator
ASL, ASLA, ASLB	Arithmetic shift of accumulator or memory left
ASR, ASRA, ASRB	Arithmetic shift of accumulator or memory right
BITA, BITB	Bit test memory with accumulator
CLR, CLRA, CLRB	Clear accumulator or memory location
CMPA, CMPB	Compare memory from accumulator
COM, COMA, COMB	Complement accumulator or memory location
DAA	Decimal adjust A accumulator
DEC, DECA, DECB	Decrement accumulator or memory location
EORA, EORB	Exclusive or memory with accumulator
EXG R1, R2	Exchange R1 with R2 (R1, R2 = A, B, CC, DP)
INC, INCA, INCB	Increment accumulator or memory location
LDA, LDB	Load accumulator from memory
LSL, LSLA, LSLB	Logical shift left accumulator or memory location
LSR, LSRA, LSRB	Logical shift right accumulator or memory location
MUL	Unsigned multiply (A × B → D)

NEG, NEGA, NEGB	Negate accumulator or memory
ORA, ORB	Or memory with accumulator
ROL, ROLA, ROLB	Rotate accumulator or memory left
ROR, RORA, RORB	Rotate accumulator or memory right
SBCA, SBCB	Subtract memory from accumulator with borrow
STA, STB	Store accumulator to memory
SUBA, SUBB	Subtract memory from accumulator
TST, TSTA, TSTB	Test accumulator or memory location
TFR R1, R2	Transfer R1 to R2 (R1, R2 = A, B, CC, DP)

NOTE : A, B, CC or DP may be pushed to (pulled from) either stack with PSHS, PSHU (PULS, PULU) instructions.

Tableau XIX-12. – 16-Bit Accumulator and Memory Instructions.

Instruction	**Description**
ADDD	Add memory to D accumulator
CMPD	Compare memory from D accumulator
EXG D, R	Exchange D with X, Y, S, U or PC
LDD	Load D accumulator from memory
SEX	Sign Extend B accumulator into A accumulator
STD	Store D accumulator to memory
SUBD	Subtract memory from D accumulator
TFR D, R	Transfer D to X, Y, S, U or PC
TFR R, D	Transfer X, Y, S, U or PC to D

NOTE : D may pushed (pulled) to either stack with PSHS, PSHU (PULS, PULU) instructions.

Tableau XIX-3. – Index/Stack Pointer Instructions.

Instruction	Description
CMPS, CMPU	Compare memory from stack pointer
CMPX, CMPY	Compare memory from index register
EXG R1, R2	Exchange D, X, Y, S, U or PC with D, X, Y, S, U or PC
LEAS, LEAU	Load effective address into stack pointer
LEAX, LEAY	Load effective address into index register
LDS, LDU	Load stack pointer from memory
LDX, LDY	Load index register from memory
PSHS	Push A, B, CC, DP, D, X, Y, U or PC onto hardware stack
PSHU	Push A, B, CC, DP, D, X, Y, X or PC onto user stack
PULS	Pull A, B, CC, DP, D, X, Y, U or PC from hardware stack
PULU	Pull A, B, CC, DP, D, X, Y, S or PC from hardware stack
STS, STU	Store stack pointer to memory
STX, STY	Store index register to memory
TFR R1, R2	Transfer D, X, Y, S, U or PC to D, X, Y, S, U or PC
ABX	Add B accumulator to X (unsigned)

Tableau XIX-4. – Branch Instructions.

Instruction	Description
	SIMPLE BRANCHES
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BMI, LBMI	Branch if minus
BPL, LBPL	Branch if plus

Instruction	Description
BCS, LBCS	Branch if carry set
BCC, LBCC	Branch if carry clear
BVS, LBVS	Branch if overflow set
BVC, LBVC	Branch if overflow clear
	SIGNED BRANCHES
BGT, LBGT	Branch if greater (signed)
BVS, LBVS	Branch if invalid twos complement result
BGE, LBGE	Branch if greater than or equal (signed)
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BLE, LBLE	Branch if less than or equal (signed)
BVC, LBVC	Branch if valid twos complement result
BLT, LBLT	Branch if less than (signed)
	UNSIGNED BRANCHES
BHI, LBHI	Branch if higher (unsigned)
BCC, LBCC	Branch if higher or same (unsigned)
BHS, LBHS	Branch if higher or same (unsigned)
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BLS, LBLS	Branch if lower or same (unsigned)
BCS, LBCS	Branch if lower (unsigned)
BLO, LBLO	Branch if lower (unsigned)
	OTHER BRANCHES
BSR, LBSR	Branch to subroutine
BRA, LBRA	Branch always
BRN, LBRN	Branch never

Tableau XIX-5. – Miscellaneous instructions.

Instruction	Description
ANDCC	AND condition code register
CWAI	AND condition code register, then wait for interrupt
NOP	No operation
ORCC	OR condition code register
JMP	Jump
JSR	Jump to subroutine
RTI	Return from interrupt
RTS	Return from subroutine
SW1, SW12, SW13	Software interrupt (absolute indirect)
SYNC	Synchronize with interrupt line

CHAPITRE XX **La programmation en Assembleur**

1. PRINCIPAUX CONSEILS

Que l'on écrive des programmes en langage évolué ou en assembleur, **il faut toujours penser à structurer au maximum son programme.**

Pour cela, la création maximum de sous-programmes réutilisables plusieurs fois par le programme principal augmente la clarté et facilite la mise au point.

Il est recommandé également de travailler sur des valeurs numériques voisines de Ø afin de simplifier les tests possibles.

Chaque sous-programme (« module ») ne doit avoir qu'un point d'entrée et un point de sortie.

Les ruptures de séquence sont déconseillées (BRA, JMP...) comme l'est GOTO en Basic.

2. PROGRAMME DE CREATION

Le programme ci-joint écrit en Basic permet la rentrée en mémoire, à partir de l'adresse &H7FØØ, des codes hexadécimaux successifs constituant le programme machine. On ne peut rentrer que 256 codes au maximum.

Une partie vérification relit tous les codes écrits en mémoire et les affiche sur l'écran. On sort de ce programme en tapant « FIN ».

Pour lancer un programme il faut ensuite faire EXEC &H7FØØ.

```
1 '              ***************************
2 '              *                           *
3 '              * CREATION - ASSEMBLEUR     *
4 '              *                           *
5 '              ***************************
6 '
7 '
8 '-------------------Initialisation-----
9 '
10 CLEAR,&H7EFF
20 DEFUSR1=&H7F00
30 N=&H7F00
32 '
34 '
35 '-----------Entree du programme------
36 '
38 '
40 X$=INPUT$(2):PRINTX$+" ";
50 D$=RIGHT$(X$,1):G$=LEFT$(X$,1)
60 IFASC(D$)<60THEND=VAL(D$)ELSED=ASC(D$
)-55
70 IFASC(G$)<60THENG=VAL(G$)ELSEG=ASC(G$
)-55
80 X=16*G+D
90 IFX$="FI"GOTO1000 ELSE POKEN,X
100 N=N+1
110 GOTO40
900 '
902 '
```

```
904 '-----------------Verification-------
906 '
908 '
1000 NF=N:PRINT:PRINT
1010 FORN=&H7F00 TO NF-1
1020   PRINTHEX$(PEEK(N))+" ";
1030   NEXTN
1040 END
1996 '
1998 '
2000 '-----------------Table--------------
2002 '
2004 '
2006 PRINT"ADRESSE DE LA TABLE : ";
2010 T$=INPUT$(6):T=VAL(T$):PRINTT$
2020 X$=INPUT$(2):PRINTX$+" ";
2030 D$=RIGHT$(X$,1):G$=LEFT$(X$,1)
2040 IFASC(D$)<60THEND=VAL(D$)ELSED=ASC(
D$)-55
2050 IFASC(G$)<60THENG=VAL(G$)ELSEG=ASC(
G$)-55
2060 X=16*G+D
2070 IFX$="FI"GOTO3000 ELSE POKET,X
2080 T=T+1
2090 GOTO2020
3000 NF=T:PRINT:PRINT
3010 FORN=VAL(T$)TO NF-1
3020   PRINTHEX$(PEEK(N))+" ";
3030   NEXTN
3040 END
```

3. *PROGRAMMES D'APPLICATIONS*

a) *Définition de la fenêtre écran*

Lors de l'étude de PUTC$ on avait mis au point un petit programme Assembleur fixant la fenêtre entre les lignes 4 et 21. En

terminant ce programme par RTS (&H39) puis en faisant : EXEC &H7FØØ, on vérifiera par LIST que la fenêtre écran est convenablement définie.

On peut sauver ce programme par :

SAVEM « FENETRE », &H7FØØ, NF, &H7FØØ (avec NF = &H7F1E)

b) *Définition du cadre en bleu (voir PUTC$)*

Ne pas oublier de finir par RTS (&H39). Même procédure que ci-dessus.

c) *Affichage d'un caractère standard (voir PUTC$)*

Ne pas oublier de finir par TRS (&H39).

d) *Affichage d'un caractère graphique utilisateur*

On utilisera le programme de création de table ci-dessous, dont on sort en tapant FIN, et qui commence par RUN 2ØØØ, ou GOTO 2ØØØ.

On tape donc d'abord la table graphique « VOITURE » puis, par RUN, on rentre le programme d'affichage et enfin on fait : EXEC &H7FØØ.

e) *Programme de rénumération des lignes de programme de 1Ø en 1Ø*

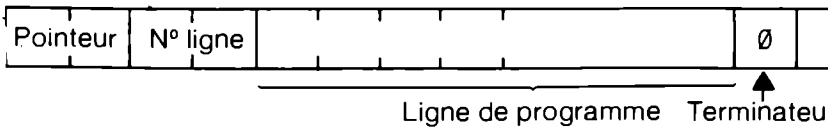
Pour bien comprendre comment fonctionne ce programme, il faut tout d'abord savoir comment un programme est rangé en mémoire vive.

– Considérons le programme suivant :

```
1Ø CLS : SCREEN 7, Ø, Ø
11 OPEN « Ø », # 1, « LPRT : (8Ø) »
15 A $ = « %      % » : B$ = « %      % »
```

4 blancs — 3 blancs

– Ce programme est rangé en RAM à partir de l'adresse &H65F5 selon le modèle ci-dessous :



Chaque ligne de programme occupe :

- 2 octets qui contiennent l'adresse du pointeur suivant,
- 2 octets qui contiennent le n° de la ligne de programme,
- X octets qui contiennent l'ensemble des instructions codées d'une ligne de programme,
- 1 octet = &HØØ de fin de ligne.
- Le premier pointeur occupe les adresses &H65F5 et &H65F6
- Le premier n° de ligne est rangé en &H65F7 et &H65F8

Dans l'exemple ci-dessous nous aurons donc :

Adresse	Valeur hexa	Commentaire
65F5 65F6	66 Ø3	Pointe vers le pointeur suivant
65F7 65F8	ØØ ØA	N° ligne = ØØØA → 10 en décimal
65F9	9D	→ Code de CLS
65FA	3A	→ Code des :
65FB 65FC	FF A4	→ Code de SCREEN
65FD	37	Code du 7
65FE	2C	Code de ,
65FF	3Ø	Code de Ø
66ØØ	2C	Code de ,
66Ø1	3Ø	Code de Ø
66Ø2	ØØ	Fin de ligne
66Ø3 66Ø4	66 1B	Pointe vers le pointeur suivant
66Ø5 66Ø6	ØØ ØB	N° de ligne = &HØØØB → 11
66Ø7	B5	Code de OPEN
66Ø8	22	Code des "
66Ø9	4F	Code de 0
660A	22	Code des "
660B	2C	Code de ,

Le pointeur de ligne contient donc toujours l'adresse du pointeur suivant et les numéros des lignes sont toujours 2 octets après le pointeur.

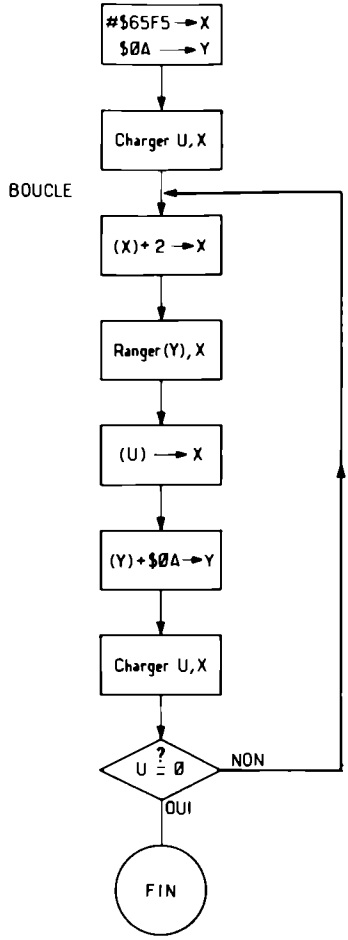


Fig. XX-1.
Organigramme de renumérotation.

Pour renuméroter les lignes de 1Ø en 1Ø, il faut donc procéder comme indiqué sur l'organigramme de la figure XX-1 :

– On charge le registre X avec l'adresse du 1er pointeur &H65F5.

– On charge le registre Y avec le premier n° de ligne dans la nouvelle numérotation, soit 1Ø (&HØA).

– On charge le registre U avec ce qui se trouve à l'adresse contenue par X, c'est-à-dire avec l'adresse du second pointeur (adressage indexé).

BOUCLE

– On incrémente le contenu de X de 2 pour pointer sur le numéro de la ligne.

– On range Y à cette adresse (indexé) ce qui revient à renuméroter la ligne.

– On remet U dans X qui contient donc à nouveau le pointeur.

– On ajoute 10 à Y → nouveau n° de ligne = 20.

– On charge à nouveau U avec ce qui se trouve à l'adresse contenue par X, c'est-à-dire le nouveau pointeur.

– On teste U = 0 (fin de programme) et sinon on recommence en BOUCLE (adressage relatif).

On obtient donc le listing d'assemblage ci-dessous :

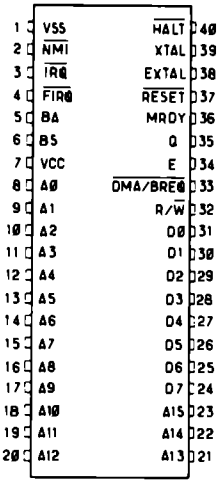
	Adresse	**Programme SOURCE**	**Programme OBJET**	**Type d'adressage**
	&H 7F 00	LDX # $ 65F5	8E 65 F5	immédiat
	&H 7F 03	LDY # $ 000A	10 8E 00 0A	immédiat
	&H 7F 07	LDU 0,X	EE 84	indexé
BOUCLE	&H 7F 09	LEAX 2,X	30 02	indexé
	&H 7F 0B	STY 0,X	10 AF 84	indexé
	&H 7F 0E	TFR U,X	1 F 31	implicite
	&H 7F 10	LEAY $ 0A,Y	31 2A	indexé
	&H 7F 12	LDU 0,X	EE 84	indexé
	&H 7F 14	BNE BOUCLE	26 F3	relatif court
	&H 7F 16	RTS	39	implicite

Il suffit ensuite de charger le programme objet ci-dessus à l'aide du programme de création-assembleur précédent et de faire ensuite EXEC &H7F00 pour renuméroter de 10 en 10 tout programme Basic présent en RAM.

Attention : Ce programme très simple ne change pas les adresses lignes de destination des instructions GOTO ou GOSUB.

On consultera à l'annexe 8 la liste des variables Basic avec leur numéro de code.

ANNEXE 1 **Le 68Ø9**



Brochage

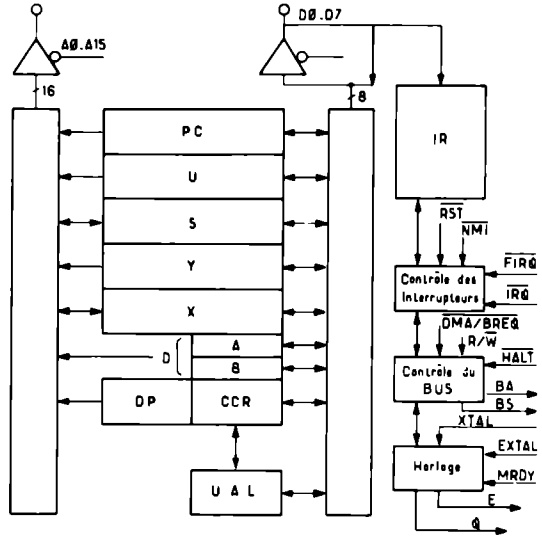


Schéma fonctionnel

E	F	H	I	N	Z	V	C

Registre Codes Condition

BA	BS	
Ø	Ø	Normal
Ø	1	Reconnaissance d'interruption
1	Ø	Reconnaissance de SYNC
1	1	Arrêt

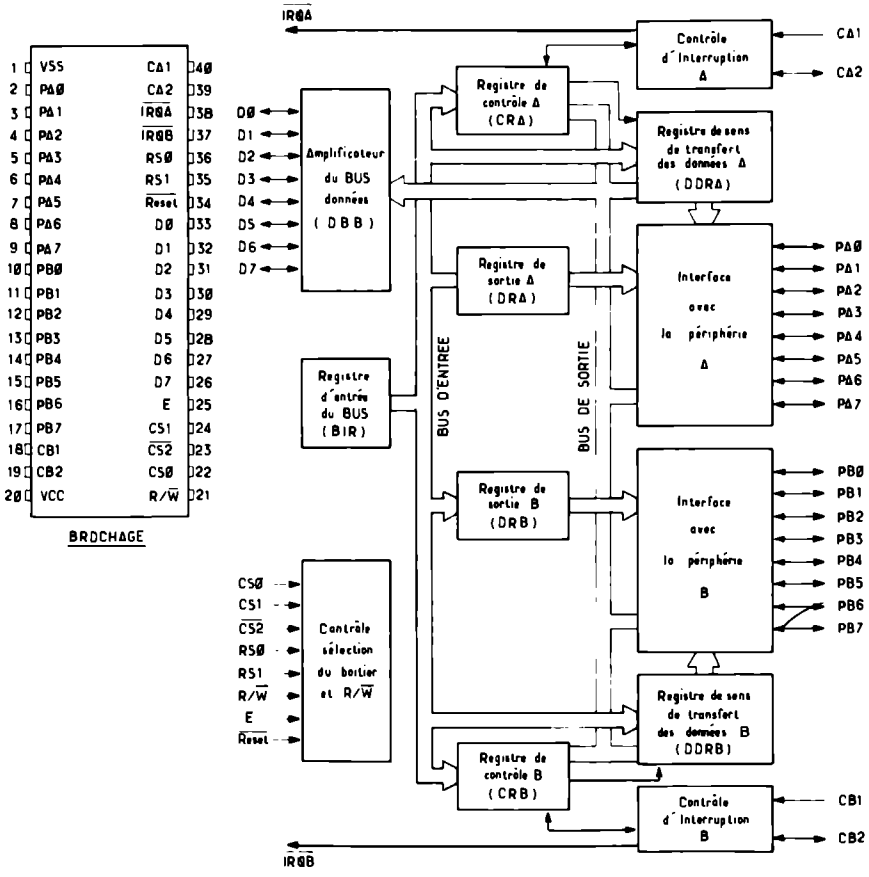
Etat du 6809

FFFE	FFFF	RESET
FFFC	FFFD	NMI
FFFA	FFFB	SWI
FFF8	FFF9	IRQ
FFF6	FFF7	FIRQ
FFF4	FFF5	SWI2
FFF2	FFF3	SWI3
FFF0	FFF1	Réservé

Pointeurs d'interruption

Note : Les demandes NMI, FIRQ et IRQ sont mémorisées sur le front descendant de Q, sauf pendant les fonctionnements en vol de cycle (exemple DMA) au cours desquels seule NMI est mémorisée.

ANNEXE 2 **Le 6821**



SCHEMA FONCTIONNEL

RS1	RSØ	CRA-2	CRB-2	Registre sélecté
Ø	Ø	1	x	ORA
Ø	Ø	Ø	x	DDRA
Ø	1	x	x	CRA*
1	Ø	x	1	ORB
1	Ø	x	Ø	DDRB
Ø	Ø	x	x	CRB

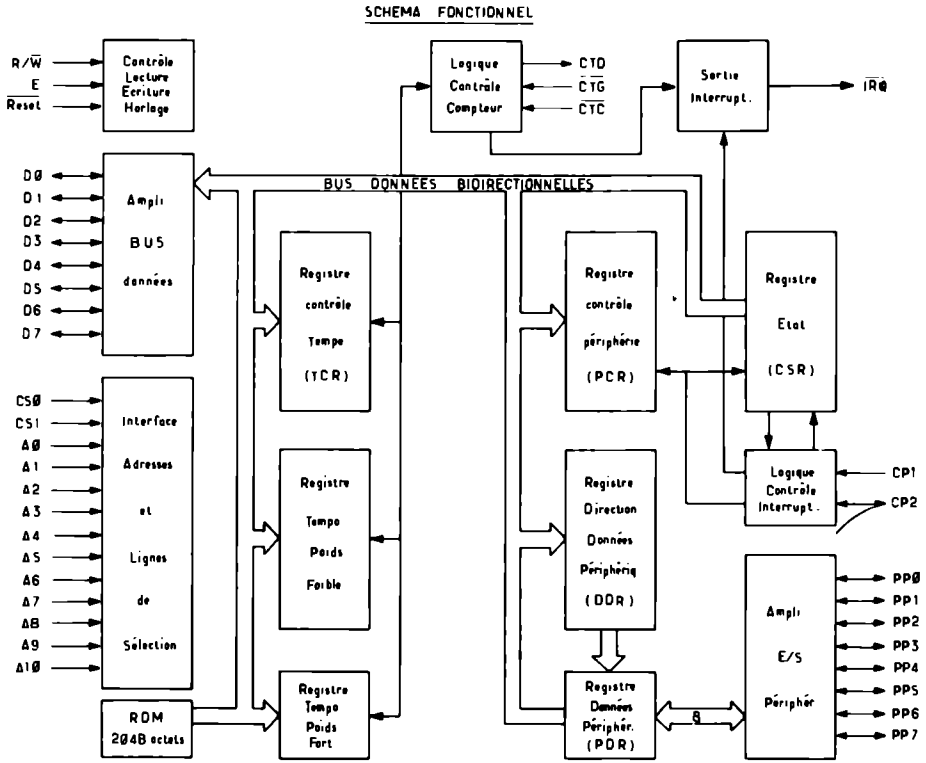
Adressage interne

7	6	5 4 3	2	1 Ø	
IRQA1	IRQA2	Contrôle de CA2	Accès à DDRA	Contrôle de CA1	CRA

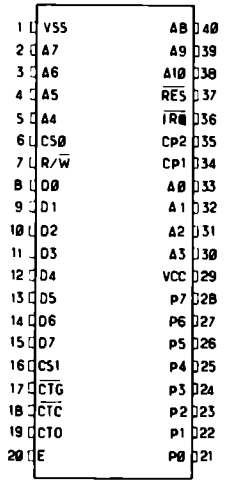
7	6	5 4 3	2	1 Ø	
IRQB1	IRQB2	Contrôle de CB2	Accès à DDRB	Contrôle de CB1	CRB

Format des mots de contrôle

ANNEXE 3 **Le 6846**



BROCHAGE



ADRESSES DES REGISTRES INTERNES

A2	A1	AO	REGISTRE SELECT.
0	0	0	CSR
0	0	1	PCR
0	1	0	DDR
0	1	1	PDR
1	0	0	CSR
1	0	1	TCR
1	1	0	Tempo Poids Fort
1	1	1	Tempo Poids Faible
X	X	X	ROM

ANNEXE 4 **Page Ø du moniteur système**

Adresse	Identificateur	Commentaires
$6ØØØ	TERMIN,25	Table des terminateurs de lignes
$6Ø19	STATUS,1	BIT7 = SEMIGRAPH, BIT6 = ROLLUP RAPIDE, BIT5 = INTERRUPT TIMER USER, BIT3 = CU.USER, BIT2 = CURSEUR, BIT1 = INCREMENT CLAVIER, BITØ = TOUCHE DEJA LUE
$6Ø1A	TABPT,1	Forme avec RANG un pointeur courant de 16 bits dans
$6Ø1B	RANG,1	la table des terminateurs de lignes
$6Ø1C	TOPTAB,1	Forme avec TOPRAN un pointeur de 16 bits sur le sommet logique de la table des terminateurs
$6Ø1D	TOPRAN,1	1re rangée de la fenêtre
$6Ø1E	BOTTAB,1	Forme avec BOTRAN un pointeur de 16 bits sur la fin logique de la table des terminateurs
$6Ø1F	BOTRAN,1	Dernière rangée de la fenêtre
$6Ø2Ø	COLN,1	
$6Ø21	IRQPT,2	Pointeur IRQ utilisateur
$6Ø23	IRQPT,2	Pointeur FIRQ utilisateur
$6Ø25	CC1PT,2	Pointeur INTERRUPT sur CC1
$6Ø27	TIMEPT,2	Pointeur INTERRUPT TIMER UTILISATEUR
$6Ø29	K7.OPC,1	Mot de code de l'opération cassette désirée
$6Ø2A	K7.STA,1	Status courant du contrôleur cassette
$6Ø2B	RS.OPC,1	Mot de code pour la liaison RS-232
$6Ø2C	RS.STA,1	Status courant de la liaison RS-232
$6Ø2D	USERAF,2	Pointeur sur le générateur de caractères usager
$6Ø2F	SWI1,2	Pointeur SWI
$6Ø31	TEMPO,2	Tempo général de l'interpréteur musical
$6Ø33	DUREE,2	Durée
$6Ø35	TIMBRE,1	Timbre
$6Ø36	OCTAVE,2	Octave
$6Ø38	FORME,1	Couleur du PLOT et du DRAW (– 8 à + 7)
$6Ø39	ATRANG,1	BIT7 = SCRFUL, BIT2 = LARGEUR, BIT1 = HAUTEUR
$6Ø3A	ATRSCR,1	BIT7 = FOND, BIT6 = FORME, BIT2 = LARGEUR, BIT1 = HAUTEUR
$6Ø3B	COLOUR,1	X.X.B.V.R.B.V.R
$6Ø3C	TELETL,1	Flag de mode Teletel et PR.STA. BIT7 = MODE PAGE, BIT6 = ECHO
$6Ø3D	PLOTX,2	Dernier plot
$6Ø3F	PLOTY,2	Dernier plot
$6Ø41	CHDRAW,1	Caractères ASCII utilisés pour PSET, LINE et BOX «CH»
$6Ø42	CURSFL,1	Flag de mouvement curseur indiquant que l'on ne pose pas de liens de ligne dans la table des terminateurs
$6Ø43	COPCHR,1	Flag indiquant que BACKSPACE et HORIZONTAL TABULATION copient le caractère courant

Adresse	Identificateur	Commentaires
$6044	BAUDS,2	Paramètre de vitesse de la liaison RS232
$6046	NOMBRE,1	Nombre de bits de transmission RS232 : t ou 2
$6047	GRCODE,1	Code déclenchant le mode graphique de l'imprimante
		PRIMITIVES DE DISQUE
$6048	DK.OPC,1	MOT DE COMMANDE : code instruction utilisé par DKCONT pour appeler une commande paramétrée par les registres qui suivent
$6049	DK.DRV,1	N° du disque sélecté
$604A	DK.TRK,2	N° de piste
$604C	DK.SEC,1	N° de secteur
$604D	DK.NUM,t	Nombre de secteurs en accès multiple
$604E	DK.STA,1	MOT D'ETAT : contient le code d'erreur signalé par C=1 en sortie de DKCONT.
$604F	DK.BUF,2	I/O BUFFER POINTER
		REGISTRES COURANTS DE PISTE
$6051	TRACKØ,2	Position courante de la tête sur le drive Ø
$6053	TRACK1,2	Position courante de la tête sur le drive 1
$6055	TRACK2,2	Position courante de la tête sur le drive 2
$6057	TRACK3,2	Position courante de la tête sur le drive 3
		AUTRES REGISTRES
$6059	SEQUCE,1	Ø = NORAML, 2 = ESCAPE, 4 = UNSEP, 6 = DEVCO2, 8 = DEVCO3
$605A	SCRPT,2	Pointeur d'écran
$605C	SAVCOL,t	Double de COLOUR
$605D	ASCII,t	Dernier caractère
$605E	KEY,1	Touche clavier
$605F	CMPTKB,1	REPEATS du clavier
$6060	STADR,2	1er octet définissant l'origine de la fenêtre
$6062	ENDDR,2	Dernier octet définissant la fin de la fenêtre
$6064	TCRSAV,1	Sauvegarde de l'ETAT courant du timer
$6065	TCTSAV,2	Sauvegarde du COMPTE courant du timer
$6067	BAUD,1	Paramètre de vitesse de transmission cassette
$6068	SAVATR,2	Sauvegarde des attributs courants d'écran
$606A	US1,1	1er caractère des séquences UNIT SEPARATOR
$606B	COMPT,t	Compteur de caractères répétés
$606C	TEMP,2	Registre temporaire
$606E	SAVEST,2	Registre de sauvegarde de S
$6070	ACCENT,t	FLAG ACCENT 2 = affich. accent ; 1 = saut des 2 octets ; 3 = cédille

ANNEXE 4 **Page Ø du moniteur système**

Adresse	Identificateur	Commentaires
$6Ø71	SS2GET,1	
$6Ø72	SS3GET,1	
$6Ø73	ABCMP,2	Compteur de la touche STOP pour avorter une lecture cassette
$6Ø75	EFCMPT,1	Flag de clignotement curseur
$6Ø76	BLOCZ,2	Bloc de deux Ø pour initialiser les registres 16 bits
$6Ø78	SCROLS,1	Flag de « smooth scroll »
$6Ø79	CHX1,2	Table des choix effectués au « menu » : 1er choix
$6Ø7B	CHRX2,2	Table des choix effectués au « menu » : 2e choix
$6Ø7D	CHRX3,2	Table des choix effectués au « menu » : 3e choix
$6Ø7F	RUNFLG,1	Flag indiquant une entrée en mode « RUN «
$6Ø8Ø	DKFLG,1	Flag indiquant la présence du contrôleur de disque
$6Ø81	STKEND,8Ø	Profondeur de la pile système
$6ØDØ	STACK,*−1	Sommet de la pile
$6ØD1	APPLIC,1	Checksum de l'application en cours
$6ØD2	DECALG,1	Décalage du lignt-pen
$6ØD3	LPBUFF,16	Buffer de saisie du light-pen
$6ØE3	NBPST,1	Nombre de pistes par face disque
$6ØE4	CPTSCT,1	Compteur de secteurs
$6ØE5	DTAB1,3	Zone de travail
$6ØE8	FTAB1,1	Fin de zone de travail DTAB1
$6ØE9	TAB2,2	ADRESS MARK CLOCKES $F5 $FE
$6ØEB	DTAB2,12	Zone de travail
$6ØF7	FTAB2,Ø	Fin de zone de travail DTAB2
		BUFFER DE LANCEMENT DU BOOTSTRAP
$62ØØ	BUFBOO,127	Buffer de chargement et de lancement du bootstrap
$627F	CHKBOO,1	Checksum du bootstrap : total 128 octets

ANNEXE 5 **Les adresses importantes**

Adresse	Identificateur	Commentaires
		ADRESSES DE L'ECRAN
$4000	STAD	1er octet
$5F40	ENDAD	Dernier octet
$4118	CURSOR	Origine de la première rangée
$4258	TELCUR	Origine de la deuxième rangée
$4140	ORGROL	Origine du ROLLUP
$4280	ORGBL	Origine du ROLLUP double hauteur
$5F18	DERCUR	Curseur sur la dernière ligne
		ADRESSES D'ENTREE-SORTIE
		6846
$E7C0	CSR	COMPOSITE STATUS REGISTER
$E7C1	CRC	Registre de contrôle du port P
$E7C2	DDRC	Registre de direction du port P
$E7C3	PRC	Registre de données du port P
$E7C5	TCR	TIMER CONTROL REGISTER
$E7C6	TMSB	TIMER M.S.B.
$E7C7	TLSB	TIMER L.S.B.
		6821 Système
$E7C8	PRA	Registre de données du port A
$E7C9	PRA	Registre de données du port B
$E7CA	CRA	Registre de contrôle du port A
$E7CB	CRB	Registre de contrôle du port B
		6821 jeux
$E7CC	PRA t	Registre de données du port A1
$E7CD	PRB1	Registre de données du port B1
$E7CE	CRA1	Registre de contrôle du port A1
$E7CF	CRB1	Registre de contrôle du port B1
		6821 RS232 et interface imprimante parallèle
$E7E0	PRA2	Registre de données du port A2
$E7E1	PRB2	Registre de données du port B2
$E7E2	CRA2	Registre de contrôle du port A2
$E7E3	CRB2	Registre de contrôle du port B2

ANNEXE 5 **Les adresses importantes**

Adresse	Identificateur	Adresse	Identificateur
	ADRESSES DES SOUS-PROGRAMMES SYSTEMES		
$E800	INIT$	$E81E	NOTE$
$E803	PUTC$	$E821	GETP$
$E806	GETC$	$E824	GETS$
$E809	KTST$	$E827	JOYS$
$E80C	DRAW$	$E82A	DKCO$
$E80F	PLOT$	$E82D	MENU$
$E812	RSCO$	$E830	KBIN$
$E815	K7CO$	$E833	CHPL$
$E818	GETL$	$E007	DKBO$
$E81B	LPIN$	$E00A	FORM$

ANNEXE 6 **Les principaux paramètres**

I. PARAMETRES MUSICAUX

	P	$30		SO	$38	MIb	RE#	$34	DOb	SI	$3C
	DO	$31	LAb	SO#	$39		MI	$35		UT	$3D
REb	DO#	$32		LA	$3A		FA	$36			
	RE	$33	SIb	LA#	$3B	SOb	FA#	$37			

II. PARAMETRES MINICASSETTE A 900 BAUDS

Tempo pour lancer la bande : 33333*30 = 1 seconde
Fréquence du 0 à 5 pulses de 4 500 Hz
Fréquence du 1 à 7 pulses de 6 300 Hz

K7.OPR	%	0	0	0	0	0	0	0	0	Ouvert en lecture
K7.RDC	%	0	0	0	0	0	0	1	0	Lecture d'un octet
K7.OPW	%	0	0	0	0	0	1	0	0	Ouvert pour écrire
K7.WRT	%	0	0	0	0	1	0	0	0	Ecrire un octet
K7.CLS	%	0	0	0	1	0	0	0	0	CLOSE
K7.ABT	%	0	1	0	0	0	0	0	0	Abort lecture cassette
K7.NRD	%	1	0	0	0	0	0	0	0	Dispositif pas prêt

III. PARAMETRES LIAISON RS232

RS.OPR	%	0	0	0	0	0	0	0		Ouvert en lecture/écriture (RS232)
RS.RDC	%	0	0	0	0	0	0	1	0	Lire un octet
RS.OPW	%	0	0	0	0	0	1	0	0	Ouvert en écriture seulement (RS232)
RS.WRC	%	0	0	0	0	1	0	0	0	Ecrire un octet
RS.CLS	%	0	0	0	1	0	0	0	0	CLOSE
RS.CPY	%	0	0	1	0	0	0	0	0	Copie graphique d'écran
RS.OPP	%	0	1	0	0	0	0	0	0	Ouvert en écriture parallèle
RS.NRD	%	1	0	0	0	0	0	0	0	Dispositif pas prêt
TXDATA	%	1	0	0	0	0	0	0	0	TRANSMIT DATA (entrée) : mark = 1, start = 0, positif
DTTRMN	%	0	1	0	0	0	0	0	0	DATA TERMINAL READY (entrée) : ready = 0, busy = 1
REQTS	%	0	0	1	0	0	0	0	0	REQUEST TOSEND (entrée) : request = 0
CLRTS	%	0	0	0	0	0	0	1	0	CLEAR TO SEND (sortie) : clear = 0
RXDATA	%	0	0	0	0	0	0	0	1	RECEIVE DATA (sortie) : mark = 1, start = 0, positif

$046A	PARAMETRE DE	110	BAUDS	$002E	PARAMETRE DE	2 400	BAUDS
$019B	–	300	–	$0015	–	4 800	–
$00CA	–	600	–	$0007	–	9 600	–
$0062	–	1 200	–				

ANNEXE 7 **Jeu de codes ASCII**

Code décimal	Code hexa	
00	00	NUL
01	01	SON
02	02	START OF TEXT
03	03	STOP (CONT/C)
04	04	END OF TRANS.
05	05	ENQUIRE
06	06	ACKNOWLEDGE
07	07	SONNETTE
08	08	BS (←)
09	09	HT (→)
10	0A	LF (↓)
11	0B	VT (↑)
12	0C	FF (RAZ)
13	0D	CR (ENTREE)
14	0E	SO (semi-graph.)
15	0F	SI (alphanum.)
16	10	DLE
17	11	DC1 (clignot. curs.)
18	12	DC2 (répétition)
19	13	DC3
20	14	DC4 (arrêt curs.)
21	15	NAK
22	16	SS2 (accent.)
23	17	ETB
24	18	CAN (CONT/X)
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	INS (INS)
29	1D	DEL (EFF)
30	1E	RS ( )
31	1F	US
32	20	Espace
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)

Code décimal	Code hexa	
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S

Code décimal	Code hexa	
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	↑
95	5F	←
96	60	—
97	61	a
98	62	b

Code décimal	Code hexa	
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q

Code décimal	Code hexa	
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	:
125	7D	}
126	7E	≈
127	7F	DEL

ANNEXE 8 **Mots réservés du Basic**

MOT	Code hexa	Page
ABS	FF82	77
AND	CC	65
ASC	FF8E	85
ATTRB	A8	47
AUTO	9A	118
BEEP	A3	100
BOX	A6	124
BOXF	A646	124
CDBL	FF93	77
CHR$	FF8F	86
CINT	FF91	77
CLEAR	AE	59,71
CLOSE	B6	104
CLS	9D	48
COLOR	A4	49,125
CONSOLE	9E	49
CONT	AC	118
COS	FF87	77
CSNG	FF92	77
CSRLIN	FFA2	54
DATA	83	40
DEF	A9	59,70
DELETE	9B	118
DIM	84	93
ELSE	8F	65
END	8Ø	
EOF	FF9Ø	104
EQV	CF	
ERL	C1	116
ERR	C2	116
ERROR	98	116
EXEC	A2	69
EXP	FF86	77
FIX	FF94	77
FN	BD	
FOR	81	67
FRE	FF83	86

MOT	Code hexa	Page
GO	87	65,69
GR$	FF99	89
HEX$	FF95	86
IF	89	65
IMP	DØ	
INKEY$	FFAØ	39
INPEN	B7	96
INPUT	FFA1	37
INSTR	FF9D	86
INT	FF81	77
LEFT$	FF9A	86
LEN	FF8B	100
LET	86	
LINE	A5	53
LIST	AD	105
LOAD	B3	105
LOCATE	9C	52
LOG	FF85	77
MERGE	B4	106
MID$	FF9C	87
MOD	D1	76
MOTOR	AØ	106
NEW	B1	119
NEXT	82	67
NOT	C5	65
OCT$	FF96	87
OFF	C3	106
ON	96	65,69
OPEN	B5	106
OR	CD	65
PEEK	FF8A	112
PEN	B8	96
PLAY	B9	100
POINT	FFA3	124
POKE	AA	112

MOT	Code hexa	Page
POS	FFA5	107
PRINT	AB	42
PSET	9F	52,124
PTRIG	FFA6	99
READ	85	40
REM	8C	
RESTORE	8A	41
RESUME	99	116
RETURN	8B	69
RIGHT$	FF9B	87
RND	FF9F	77
RUN	88	119
SAVE	B2	107
SCREEN	FFA4	52,54
SGN	FF8Ø	78
SIN	FF88	78
SKIPF	A1	107
SPC(	BE	42
SQR	FF84	78
STEP	C6	67
STICK	FF97	100
STOP	8E	
STRIG	FF98	100
STR$	FF8C	87
SUB	BC	69
TAB(	BA	42
TAN	FF89	78
THEN	C4	65
TO	BB	65,67
TROFF	91	119
TRON	90	119

MOT	Code hexa	Page
UNMASK	A7	48
USING	BF	43
USR	CØ	70
VAL	FF8D	87
VARPTR	FF9E	115
WAIT	97	107
WEND	BØ	
WHILE	AF	
XOR	CE	65
CODES SPECIAUX		
DEFINT	93	75
DEFDBL	95	76
DEFSNG	94	76
DEFSTR	92	76
+	C7	76
=	D4	65,76
*	C9	76
/	CA	76
-	C8	76
<	D5	65,76
>	D3	65,76

ANNEXE 9 **Codes d'erreurs**

Code	numéro	
AO	**52**	Fichier déjà ouvert (Already Open)
BD	**58**	Mauvaises données dans le fichier (Bad Data)
BS	**9**	Indice en dehors des limites (Subscript out of range)
CN	**17**	Suite d'exécution impossible (Can't continue) – arrêt sur erreur – modification du programme après une interruption par **CNT - C**
DD	**10**	Définition multiple ou définition d'un tableau après l'avoir utilisé avec un dimensionnement implicite (Redimensioned array)
DS	**56**	Commande directe dans un fichier **ASCII** en cours de chargement (Direct Statement)
DU	**60**	Périphérique non disponible (Device unavailable)
FC	**5**	Argument incorrect dans l'appel d'une fonction ou de certaines instructions (Illegal function call)
FM	**51**	Mode d'accès au fichier incorrect (Bad file mode)
FN	**23**	Instruction **FOR** sans instruction **NEXT** (For without Next)
ID	**12**	Instruction interdite en mode immédiat (Illegal Direct)
IE	**54**	Tentative de lecture d'un fichier après la fin de fichier. Pour éviter cette erreur, utiliser la fonction **EOF** (Input past End)
IO	**53**	Erreur d'entrée/sortie (Input/Output)
IU	**59**	Périphérique en cours de fonctionnement (Device in use)
LS	**15**	Chaîne de caractères trop longue (String too long)
MO	**22**	Opérande manquant (Missing Operand)
NF	**1**	Instruction **NEXT** sans instruction **FOR** ou **NEXT** avec variable ne correspondant pas à l'instruction **FOR** exécutée (Next without For)
NO	**57**	Fichier non ouvert (Not Open)
NR	**19**	Instruction **RESUME** manquante dans le sous-programme de traitement des erreurs (No Resume)
NU	**50**	Numéro de fichier incorrect (Not in use)
OM	**7**	Dépassement de capacité en mémoire centrale (Out of Memory)
OD	**4**	Instruction **READ** alors que les données des instructions **DATA** sont épuisées (Out of Data)
OS	**14**	Manque de place pour les chaînes (Out of string Space)
OV	**6**	Dépassement de capacité par valeur supérieure (Overflow)
PP	**61**	Programme protégé (Protected Program)
RE	**20**	Instruction **RESUME** alors qu'il n'y a pas eu d'erreur détectée (Resume without Error)

Code	numéro	
RG	**3**	Instruction **RETURN** alors que **GOSUB** n'a pas été exécuté (Return without Gosub)
SN	**2**	Erreur de syntaxe (Syntax error)
ST	**16**	Expression sur chaîne trop complexe (String formula Too complex)
TM	**13**	Non correspondance de type (Type Mismatch)
UE	**21**	Erreur non définie (Undefined Error)
UF	**18**	Fonction utilisateur non définie (Undefined user Function)
UL	**8**	Numéro de ligne inexistant (Undefined Line)
/0	**11**	Division par zéro

ANNEXE 10 **Le jeu d'instructions du 6809**

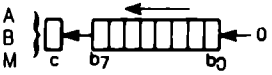
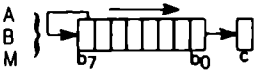
Instructions de branchement

Instruction	Forms	Addressing Mode			Description	5	3	2	1	0
		Relative								
		Op	~	#		H	N	Z	V	C
BCC	BCC LBCC	24 10 24	3 5(6)	2 4	Branch C = 0 Long Branch C = 0	● ●	● ●	● ●	● ●	● ●
BCS	BCS LBCS	25 10 25	3 5(6)	2 4	Branch C = 1 Long Branch C = 1	● ●	● ●	● ●	● ●	● ●
BEQ	BEQ LBEQ	27 10 27	3 5(6)	2 4	Branch Z = 0 Long Branch Z = 0	● ●	● ●	● ●	● ●	● ●
BGE	BGE LBGE	2C 10 2C	3 5(6)	2 4	Branch ≥ Zero Long Branch ≥ Zero	● ●	● ●	● ●	● ●	● ●
BGT	BGT LBGT	2E 10 2E	3 5(6)	2 4	Branch > Zero Long Branch > Zero	● ●	● ●	● ●	● ●	● ●
BHI	BHI LBHI	22 10 22	3 5(6)	2 4	Branch Higher Long Branch Higher	● ●	● ●	● ●	● ●	● ●
BHS	BHS LBHS	24 10 24	3 5(6)	2 4	Branch Higher or Same Long Branch Higher or Same	● ●	● ●	● ●	● ●	● ●
BLE	BLE LBLE	2F 10 2F	3 5(6)	2 4	Branch ≤ Zero Long Branch ≤ Zero	● ●	● ●	● ●	● ●	● ●
BLO	BLO LBLO	25 10 25	3 5(6)	2 4	Branch Lower Long Branch Lower	● ●	● ●	● ●	● ●	● ●

Instruction	Forms	Addressing Mode			Description	5	3	2	1	0
		Relative								
		Op	~	#		H	N	Z	V	C
BLS	BLS LBLS	23 10 23	3 5(6)	2 4	Branch Lower or Same Long Branch Lower or Same	• •	• •	• •	• •	• •
BLT	BLT LBLT	2D 10 2D	3 5(6)	2 4	Branch < Zero Long Branch < Zero	• •	• •	• •	• •	• •
BMI	BMI LBMI	2B 10 2B	3 5(6)	2 4	Branch Minus Long Branch Minus	• •	• •	• •	• •	• •
BNE	BNE LBNE	26 10 26	3 5(6)	2 4	Branche Z ≠ 0 Long Branch Z ≠ 0	• •	• •	• •	• •	• •
BPL	BPL LBPL	2A 10 2A	3 5(6)	3 4	Branch Plus Long Branch Plus	• •	• •	• •	• •	• •
BRA	BRA LBRA	20 16	3 5	2 3	Branch Always Long Branch Always	• •	• •	• •	• •	• •
BRN	BRN LBRN	21 10 21	3 5	2 4	Branch Never Long Branch Never	• •	• •	• •	• •	• •
BSR	BSR LBSR	8D 17	7 9	2 3	Branch to Subroutine Long Branch to Subroutine	• •	• •	• •	• •	• •
BVC	BVC LBVC	28 10 28	3 5(5)	2 4	Branch V = 0 Long Branch V = 0	• •	• •	• •	• •	• •
BVS	BVS LBVS	29 10 29	3 5(6)	2 4	Branch V = 1 Long Branch V = 1	• •	• •	• •	• •	• •

ANNEXE 10 **Le jeu d'instructions du 68Ø9**

Instruction	Forme	Addressing modes											
		Immediate			Direct			Indexed			Extended		
		Op	~	#	Op	~	#	Op	~	#	Op	~	#
ABX													
ADC	ADCA	89	2	2	99	4	2	A9	4+	2+	B9	5	3
	ADCB	C9	2	2	D9	4	2	E9	4+	2+	F9	5	3
ADD	ADDA	8B	2	2	9B	4	2	AB	4+	2+	BB	5	3
	ADDB	CB	2	2	DB	4	2	EB	4+	2+	FB	5	3
	ADDD	C3	4	3	D3	6	2	E33	6+	2+	F3	7	3
AND	ANDA	84	2	2	94	4	2	A4	4+	2+	B4	5	3
	ANDB	C4	2	2	D4	4	2	E4	4+	2+	F4	5	3
	ANDCC	1C	3	2									
ASL	ASLA												
	ASLB												
	ASL				08	6	2	68	6+	2+	78	7	3
ASR	ASRA												
	ASRB												
	ASR				07	6	2	67	6+	2+	77	7	3
BIT	BITA	85	2	2	95	4	2	A5	4+	2+	B5	5	3
	BITB	C5	2	2	D5	4	2	E5	4+	2+	F5	5	3
CLR	CLRA												
	CLRB												
	CLR				0F	6	2	6F	6+	2+	7F	7	3
CMP	CMPA	81	2	2	91	4	2	A1	4+	2+	B1	5	3
	CMPB	C1	2	2	D1	4	2	E1	4+	2+	F1	5	3
	CMPD	10	5	4	10	7	3	10	7+	3+	10	8	4
		83			93			A3			B3		
	CMPS	11	5	4	11	7	3	11	7+	3+	11	8	4
		8C			9C			AC			BC		
	CMPU	11	5	4	11	7	3	11	7+	3+	11	8	4
		83			93			A3			B3		
	CMPX	8C	4	3	9C	6	2	AC	6+	2+	BC	7	3
	CMPY	10	5	4	10	7	3	10	7+	3+	10	8	4
		8C			9C			AC			BC		
COM	COMA												
	COMB												
	COM				03	6	2	63	6+	2+	73	7	3

Inherent			Description	5	3	2	1	0
Op	~	#		H	N	Z	V	C
3A	3	1	B + X → X (Unsigned)	●	●	●	●	●
			A + M + C → A B + M + C → B	↕ ↕	↕ ↕	↕ ↕	↕ ↕	↕ ↕
			A + M → A B + M → B D + M:M + 1 → D	↕ ↕ ●	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕
			A M → A B M → B CC IMM → CC	● ●	↕ ↕	↕ ↕	0 0	● ● 7
48 58	2 2	1 1	A B M 	8 8 8	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕
47 57	2 2	1 1	A B M 	8 8 8	↕ ↕ ↕	↕ ↕ ↕	● ● ●	↕ ↕ ↕
			Bit Test A (M A) Bit Test B (M B)	● ●	↕ ↕	↕ ↕	0 0	● ●
4F 5F	2 2	t 1	0 → A 0 → B 0 → M	● ● ●	0 0 0	↕ 1 1	0 0 0	0 0 0
			Compare M from A Compare M from B Compare M:M + t from D Compare M:M + t from S Compare M:M + 1 from U Compare M:M + t from X Compare M:M + t from Y	8 8 ● ● ● ● ●	↕ ↕ ↕ ↕ ↕ ↕ ↕	↕ ↕ ↕ ↕ ↕ ↕ ↕	↕ ↕ ↕ ↕ ↕ ↕ ↕	↕ ↕ ↕ ↕ ↕ ↕ ↕
43 53	2 2	t t	A → A B → B M → M	● ● ●	↕ ↕ ↕	↕ ↕ ↕	0 0 0	1 t 1

ANNEXE 10 **Le jeu d'instructions du 68Ø9**

Instruction	Forms	Addressing modes											
		Immediate			Direct			Indexed			Extended		
		Op	~	#	Op	~	#	Op	~	#	Op	~	#
CWAI		3C	≥20	2									
DAA													
DEC	DECA DECB DEC				0A	6	2	6A	6+	2+	7A	7	3
EOR	EORA EORB	88 C8	2 2	2 2	98 D8	4 4	2 2	A8 E8	4+ 4+	2+ 2+	B8 F8	5 5	3 3
EXG	R1, R2	1E	8	2									
INC	INCA INCB INC				0C	6	2	6C	6+	2+	7C	7	3
JMP					0E	3	2	6E	3+	2+	7E	4	3
JSR					9D	7	2	AD	7+	2+	BD	8	3
LD	LDA LDB LDD LDS LDU LDX LDY	86 C6 CC 10 CE CE 8E 10 8E	2 2 3 4 3 3 4	2 2 3 4 3 3 4	96 D6 DC 10 DE DE 9E 10 9E	4 4 5 6 5 5 6	2 2 2 3 2 2 3	A6 E6 EC 10 EE EE AE 10 AE	4+ 4+ 5+ 6+ 5+ 5+ 6+	2+ 2+ 2+ 3+ 2+ 2+ 3+	B6 F6 FC 10 FE FE BE 10 BE	5 5 6 7 6 6 7	3 3 3 4 3 3 4
LEA	LEAS LEAU LEAX LEAY							32 33 30 31	4+ 4+ 4+ 4+	2+ 2+ 2+ 2+			

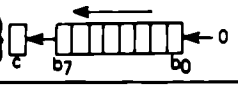
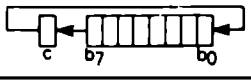
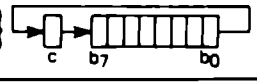
Inherent			Description	5	3	2	1	0
Op	~	#		H	N	Z	V	C
			CC IMM → CC Wait for Interrupt					7
19	2	1	Decimal Adjust A	●	↑	↑	0	1
4A	2	1	A 1 → A	●	↑	↑	↑	●
5A	2	1	B 1 → B	●	↑	↑	↑	●
			M 1 → M	●	↑	↑	↑	●
			A ∀ M → A	●	↑	↑	0	●
			B ∀ M → A	●	↑	↑	0	●
			R1 → R2(2)	●	●	●	●	●
4C	2	1	A + 1 → A	●	↑	↑	↑	●
5C	2	1	B + 1 → B	●	↑	↑	↑	●
			M + 1 → M	●	↑	↑	↑	●
			EA(3) → PC	●	●	●	●	●
			Jump to Subroutine	●	●	●	●	●
			M → A	●	↑	↑	0	●
			M → B	●	↑	↑	0	●
			M:M + 1 → D	●	↑	↑	0	●
			M:M + 1 → S	●	↑	↑	0	●
			M:M + 1 → U	●	↑	↑	0	●
			M:M + 1 → X	●	↑	↑	0	●
			M:M + 1 → Y	●	↑	↑	0	●
			EA(3) → S	●	●	●	●	●
			EA(3) → U	●	●	●	●	●
			EA(3) → X	●	●	↑	●	●
			EA(3) → Y	●	●	↑	●	●

Legend :

OP Operation Code (Hexa)
~ Number of MPU Cycles
# Number of Program Bytes
+ Arithmetic Plus
- Arithmetic Minus
● Multiply
M Complement of M
→ Transfer Into
H Half-carry (from bit 3)
N Negative (sign bit)
Z Zero (Reset)
V Overflow, 2's complement
C Carry from ALU
↑ Test and set if true, cleared otherwise
● Not Affected
CC Condition Code Register
: Concatenation
V Logical or
Logical and
∀ Logical Exclusive or

ANNEXE 10 **Le jeu d'instructions du 68Ø9**

Instruction	Forms	Addressing modes											
		Immediate			Direct			Indexed			Extended		
		Op	~	#	Op	~	#	Op	~	#	Op	~	#
LSL	LSLA LSLB LSL				08	6	2	68	6+	2+	78	7	3
LSR	LSRA LSRB LSR				04	6	2	64	6+	2+	74		3
MUL													
NEG	NEGA NEGB NEG				00	6	2	60	6+	2+	70	7	3
NOP													
OR	ORA ORB ORCC	8A CA 1A	2 2 3	2 2 2	9A DA	4 4	2 2	AA EA	4+ +	2+ 2+	BA FA	5 5	3 3
PSH	PSHS PSHU	34 36	5+4 5+4	2 2									
PUL	PULS PULU	35 37	5+4 5+4	2 2									
ROL	ROLA ROLB ROL				09	6	2	69	6+	2+	79	7	3
ROR	RORA RORB ROR				06	6	2	66	6+	2+	76	7	3
RTI													
RTS													
SBC	SBCA SBCB	82 C2	2 2	2 2	92 D2	4 4	2 2	A2 E2	4+ 4+	2+ 2+	B2 F2	5 5	3 3
SEX													

Inherent			Description	5	3	2	1	0
Op	~	#		H	N	Z	V	C
48 58	2 2	1 1	A B M 	● ● ●	↑ ↑ ↑	↑ ↑ ↑	↑ ↑ ↑	↑ ↑ ↑
44 54	2 2	1 1	A B M	● ● ●	0 0 0	↑ ↑ ↑	● ● ●	↑ ↑ ↑
3D	11	1	A x B → D (Unsigned)	●	●	↑	●	9
40 50	2 2	1 1	A + 1 → A B + 1 ↑ B M + 1 → M	8 8 8	↑ ↑ ↑	↑ ↑ ↑	↑ ↑ ↑	↑ ↑ ↑
12	2	1	No Operation	●	●	●	●	●
			A V M → A B V M → B CC V IMM → CC	● ●	↑ ↑	↑ ↑	0 0 7	● ●
			Push Registers on S Stack Push Registers on U Stack	● ●	● ●	● ●	● ●	● ●
			Pull Registers from S Stack Pull Registers from U Stack	● ●	● ●	● ●	● ●	● ●
49 59	2 2	1 1	A B M 	● ● ●	↑ ↑ ↑	↑ ↑ ↑	↑ ↑ ↑	↑ ↑ ↑
46 56	2 2	1 1	A B M 	● ● ●	↑ ↑ ↑	↑ ↑ ↑	● ● ●	↑ ↑ ↑
3B	6/15	1	Return from Interrupt					7
39	5	1	Return from Subroutine	●	●	●	●	●
			A - M - C → A B - M - C → B	8 8	↑ ↑	↑ ↑	↑ ↑	↑ ↑
1D	2	1	Sign Extend B into A	●	↑	↑	0	●

ANNEXE 10 **Le jeu d'instructions du 68Ø9**

Instruction	Forms	Addressing modes											
		Immediate			Direct			Indexed			Extended		
		Op	~	#	Op	~	#	Op	~	#	Op	~	#
ST	STA				97	4	2	A7	4+	2+	B7	5	3
	STB				D7	4	2	E7	4+	2+	F7	5	3
	STD				DD	5	2	ED	5+	2+	FD	6	3
	STS				10	6	3	10	6+	3+	10	7	4
					DF			EF			FF		
	STU				DF	5	2	EF	5+	2+	FF	6	3
	STX				9F	5	2	AF	5+	2+	BF	6	3
	STY				10	6	3	10			10	7	4
					9F			AF	6+	3+	BF		
SUB	SUBA	80	2	2	90	4	2	A0	4+	2+	B0	5	3
	SUBB	C0	2	2	D0	4	2	E0	4+	2+	F0	5	3
	SUBD	83	4	3	93	6	2	A3	6+	2+	B3	7	3
SWI	SWI(6)												
	SWI2(6)												
	SWI3(6)												
SYNC													
TFR	R1, R2	1F	6	2									
TST	TSTA												
	TSTB												
	TST				0D	6	2	6D	6+	2+	7D	7	3

Inherent			Description	5	3	2	1	0
Op	~	#		H	N	Z	V	C
			A → M	●	↑	↑	0	●
			B → M	●	↑	↑	0	●
			D → M:M + 1	●	↑	↑	0	●
			S → M:M + 1	●	↑	↑	0	●
			U → M:M + 1	●	↑	↑	0	●
			X → M:M + 1	●	↑	↑	0	●
			Y → M:M + 1	●	↑	↑	0	●
			A – M → A	8	↑	↑	↑	↑
			B – M → B	8	↑	↑	↑	↑
			D – M:M + 1 → D	●	↑	↑	↑	↑
3F	19	1	Software Interrupt 1	●	●	●	●	●
10 3F	20	2	Software Interrupt 2	●	●	●	●	●
11 3F	20	1	Software Interrupt 3	●	●	●	●	●
13	≥ 4	1	Synchronize to Interrupt	●	●	●	●	●
			R1 → R2 (2)	●	●	●	●	●
4D	2	1	Test A	●	↑	↑	0	●
5D	2	1	Test B	●	↑	↑	0	●
			Test M	●	↑	↑	0	●

Notes :

1. This column gives a base cycle and byte count. To obtain total count, add the values obtained from the INDEXED ADDRESSING MODE table, in Appendix F.
2. R1 and R2 may be any pair of 8 bit or any pair of 16 bit registers.
   The 8 bit registers are : A, B, CC, DP
   The 16 bit registers are : X, Y, U, S, D, PC
3. EA is the effective address.
4. The PSH and PUL instructions require 5 cycles plus 1 cycle for each byte pushed or pulled.
5. 5(6) means : 5 cycles if branch not taken, 6 cycles if taken (Branch instructions).
6. SWI sets I and F bits. SWI2 and SWI3 do not affect I and F.
7. Conditions Codes set as a diect result of the instruction.
8. Value of half-carry flag is undefined.
9. Special Case – Carry set if b7 is SET.

ANNEXE 11 **Modem pour TO 7**

Ce modem s'interface d'un côté au bus interne du micro-ordinateur et de l'autre à la ligne du réseau téléphonique commuté.

Il transmet les informations binaires provenant du terminal en mode synchrone et fonctionne en modulation par déplacement de fréquence.

Selon l'application, la transmission s'effectue :
- en 75/1 200 bits/s duplex intégral dans le cas d'une liaison avec une banque de données ;
- en 1 200/1 200 bits/s alternat pour liaison entre ordinateurs.

Ce modem est conforme aux recommandations de l'avis V23 du CCITT.

	Fonctionnement 75/1 200	**Fonctionnement 1 200**
Débit d'information	1 200 bits/s rec/ 75 bits/s em	1 200 bits/s
Rapidité de modulation	1 200 bauds rec/ 75 bauds em	1 200 bauds
Mode de transmission	asynchrone	asynchrone
Type de modulation	déplacement de fréquence	déplacement de fréquence
Procédure de transmission	duplex intégral	alternat
Liaison à la voie de transmission :		
• Support de transmission	RC	RC
• Impédance aux accès	600 Ω symétrique	600 Ω symétrique
• Niveau émission	– 2 et – 10 dBm	– 2 et – 10 dBm
• Niveau réception	0 à – 43 dBm	0 à – 43 dBm
• Régulation du courant continu de ligne	automatique	automatique
Alimentation	+ 5 V ± 12 V	+ 5 V/ ± 12 V
Consommation	< 1,5 W	< 1,5 W
Température fonctionnement	5 °C à + 45 °C	5 °C à + 45 °C
Température stockage	– 40 °C à + 70 °C	– 40 °C à + 70 °C
Humidité relative	⩽ 80 %	⩽ 80 %
Dimensions : H x L x P (mm)	70 x 70 x 50	70 x 70 x 50

. Achevé d'imprimer sur les presses de l'imprimerie Marcel Bon 70001 Vesoul
Dépôt légal novembre 1983 · N° d'éditeur : 404 - N° d'imprimeur : 2724